defect_correction

March 20, 2021

1 Defects correction algorithm using point cloud data

1.1 Introduction

This program is the testing playground for developing the defect correction algorithm. Things to explore here: - Basic point cloud visualization by python - Point cloud segmentation - Contour extraction by DBSCAN clustering - Different method for polygon infill - polygon and line intersection, segmentation - Zig-zag path generation

1.2 Set up

[1]:
```
# Scikit-Learn  0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

import pandas as pd
import open3d as o3d
import numpy as np
import math

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
# %matplotlib auto
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

```
# Where to save the figures
PROJECT_ROOT_DIR = "."
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images")
DATA_PATH = os.path.join(PROJECT_ROOT_DIR, "data")
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

1.2.1 1. Point cloud processing by Open3D library

step one, obtain, convert the point cloud data into desired format The accepted format for open3d library are ply, pcd, txt, ect. The best practice is using the **pcd** file format as default, as the it is consistent with other program.

1.2.2 File IO by open3d

[2]:
```
## File IO
print("Testing IO for point cloud ...")

##--------------read point cloud data, save to open3d object--------------
pcd = o3d.io.read_point_cloud(os.path.join(DATA_PATH, "Point_cloud.pcd"))
# pcd = o3d.io.read_point_cloud(os.path.join(DATA_PATH, "Y1_seg.pcd"))
print(pcd)

## write the point cloud into a pcd file
# o3d.io.write_point_cloud(os.path.join(DATA_PATH, "copy_of_points.pcd"), pcd)
```

```
Testing IO for point cloud …
geometry::PointCloud with 98371 points.
```

1.2.3 Point cloud visualizaiton by open3d

[3]:
```
# normal visualization, this will prompt a window
o3d.visualization.draw_geometries([pcd])
```

```python
# for advanced visualization from jupyter lab
from open3d import JVisualizer

visualizer = JVisualizer()
visualizer.add_geometry(pcd)
visualizer.show()
```

```
JVisualizer with 1 geometries
```

1.2.4 voxel grid downsampling

[4]:
```python
print("Downsample the point cloud with a voxel of 0.5")
downsample_pcd = pcd.voxel_down_sample(voxel_size=0.5)

o3d.visualization.draw_geometries([downsample_pcd])
```

```
Downsample the point cloud with a voxel of 0.5
```

1.3 2. Point cloud visualizaiton by matplotlib

[5]:
```python
# ---- using numpy to print all the points -------
points_np_array = np.asarray(pcd.points)
print("the points are: \n")
print(points_np_array)
#------extract x,y,z value of the points-----
x = points_np_array[:,0]
y = points_np_array[:,1]
z = points_np_array[:,2]
print("the z value extracted: \n" )
print (z)

# --------mean height-------------------
z_mean = np.mean(z)
z_mean = z_mean + 0.17
print("Z mean: " + str(z_mean))
```

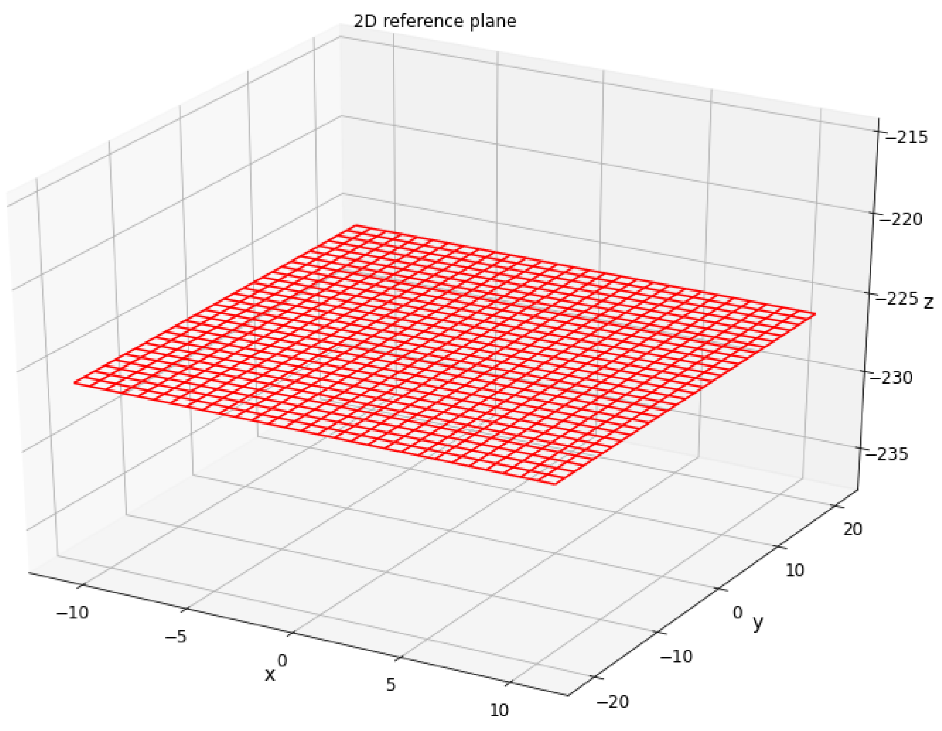
```
the points are:

[[ -10.13915588  -19.27032314 -226.7965633 ]
 [ -10.13968095  -19.19431511 -226.762755  ]
 [ -10.14048379  -19.1195331  -226.7369091 ]
 …
 [  10.17310155   19.43268697 -227.109935  ]
 [  10.17420343   19.51052893 -227.1563463 ]
 [  10.17346534   19.58841296 -227.1986699 ]]
```

```
the z value extracted:

[-226.7965633 -226.762755  -226.7369091 … -227.109935  -227.1563463
 -227.1986699]
Z mean: -225.9864015660957
```

1.4 Calculate the plane for segmentation

[6]:
```python
# Plot for a three-dimensional Plane
def z_function(x, y):
    # ax+by+cz = d ---> z = -(ax+by-d)/c
    return z_mean + 0 * x + 0 * y


x_plane = np.linspace(-11, 11, 30)
y_plane = np.linspace(-21, 21, 30)

X, Y = np.meshgrid(x_plane, y_plane)
Z = z_function(X, Y)

plt.figure(figsize=(14, 10))
ax = plt.axes(projection='3d')
# ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.plot_wireframe(X, Y, Z, color='red')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.title('2D reference plane')

# fig = plt.figure()
# ax = plt.axes(projection="3d")
# def z_function(x, y):
#     return np.sin(np.sqrt(x ** 2 + y ** 2))

# x = np.linspace(-6, 6, 30)
# y = np.linspace(-6, 6, 30)

# X, Y = np.meshgrid(x, y)
# Z = z_function(X, Y)

# fig = plt.figure()
# ax = plt.axes(projection="3d")
# ax.plot_wireframe(X, Y, Z, color='green')
# ax.set_xlabel('x')
# ax.set_ylabel('y')
# ax.set_zlabel('z')
# plt.show()

# ax = plt.axes(projection='3d')
# ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
#                 cmap='winter', edgecolor='none')
# ax.set_title('surface');
```

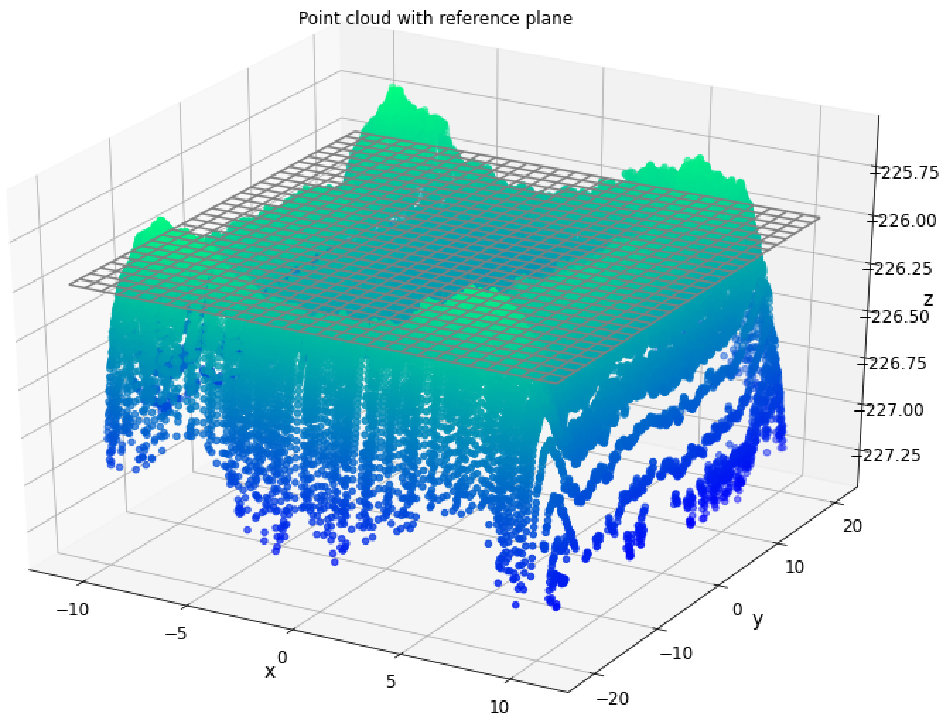
```
[6]: Text(0.5, 0.92, '2D reference plane')
```



```
[7]: # this will make the plot interactive
     # %matplotlib auto

     plt.figure(figsize=(14, 10))
     # plt.figure()
     ax = plt.axes(projection='3d')

     # Data for a three-dimensional line
     # zline = np.linspace(0, 15, 1000)
     # xline = np.sin(zline)
     # yline = np.cos(zline)
     # ax.plot3D(xline, yline, zline, 'gray')
```

```
ax.plot_wireframe(X, Y, Z, color='gray')
# Data for three-dimensional scattered points, extracted from numpy array
ax.scatter3D(x, y, z, c=z, cmap='winter'); # other cmap options: winter, Greens
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.title('Point cloud with reference plane')
```

[7]:
```
Text(0.5, 0.92, 'Point cloud with reference plane')
```



1.4.1 1.4 Point cloud visualizaiton using plotpy

[8]:
```
# # For plotting
# import plotly.io as pio
# import plotly.graph_objects as go
# import chart_studio.plotly as py

# points_np_array_downsampled = np.asarray(downsample_pcd.points)
# print("the points are: \n")
# print(points_np_array)
# #------extract x value of the points-----
# x_down = points_np_array_downsampled[:,0]
# y_down = points_np_array_downsampled[:,1]
# z_down = points_np_array_downsampled[:,2]



# layout = go.Layout(
#     title='Parametric Plot',
#     scene=dict(
#         xaxis=dict(
#             gridcolor='rgb(255, 255, 255)',
#             zerolinecolor='rgb(255, 255, 255)',
#             showbackground=True,
#             backgroundcolor='rgb(230, 230,230)'
#         ),
#         yaxis=dict(
#             gridcolor='rgb(255, 255, 255)',
#             zerolinecolor='rgb(255, 255, 255)',
#             showbackground=True,
#             backgroundcolor='rgb(230, 230,230)'
#         ),
#         zaxis=dict(
#             gridcolor='rgb(255, 255, 255)',
#             zerolinecolor='rgb(255, 255, 255)',
#             showbackground=True,
#             backgroundcolor='rgb(230, 230,230)'
#         )
#     )
# )


# fig = go.Figure(data=[go.Scatter3d(x=x_down, y=y_down, z=z_down,
#                                     mode='markers')], layout=layout)

# py.iplot(fig, filename='jupyter-parametric_plot')
# fig.show()
```

1.5 Segmentation: Remove the points below the plane

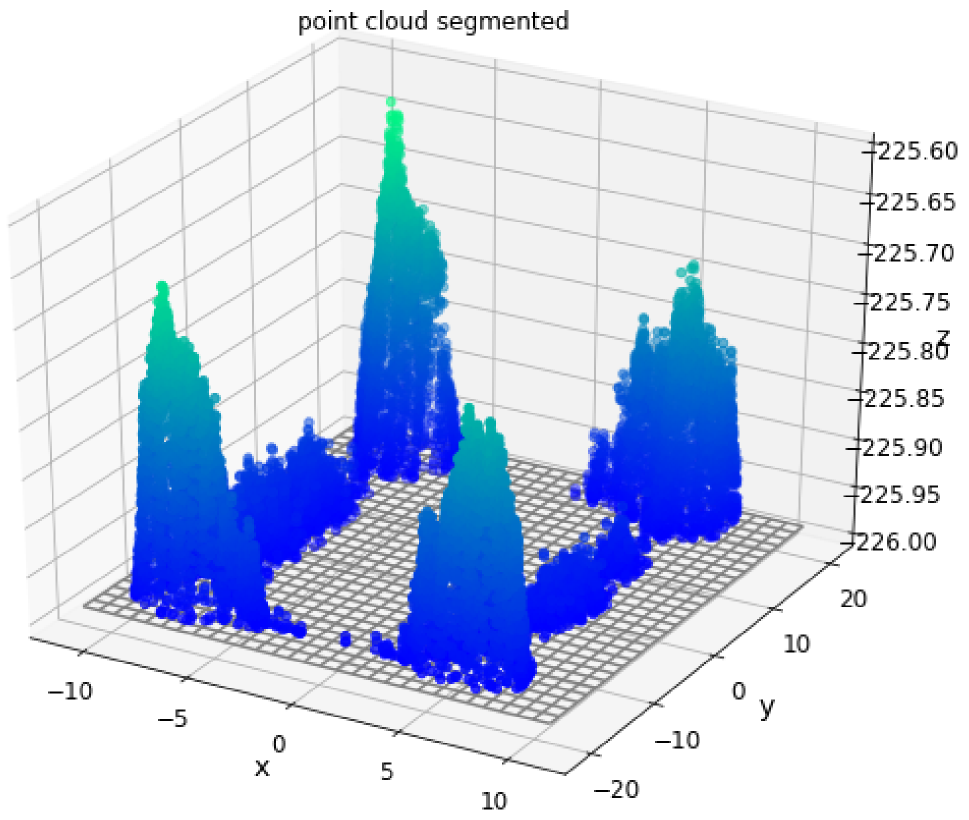
[9]:
```
points_segmented = points_np_array[np.where(points_np_array[:,2] > z_mean)]

print("the point cloud segmented: ")
print(points_segmented)
```

```
the point cloud segmented:
```

```
[[  -9.56302014  -17.74887974 -225.9817953 ]
 [  -9.56279359  -17.67781856 -225.9829139 ]
 [  -9.5634848   -17.60777515 -225.9829742 ]
 …
 [   8.66727035   17.74265523 -225.9529622 ]
 [   8.66658094   17.81269927 -225.9530025 ]
 [   8.66716664   17.88435272 -225.9691718 ]]
```

[10]:
```
x_seg = points_segmented[:,0]
y_seg = points_segmented[:,1]
z_seg = points_segmented[:,2]




plt.figure(figsize=(10, 8))
# plt.figure()
ax = plt.axes(projection='3d')

ax.plot_wireframe(X, Y, Z, color='gray')
# Data for three-dimensional scattered points, extracted from numpy array
ax.scatter3D(x_seg, y_seg, z_seg, c=z_seg, cmap='winter'); # other cmap options:
 ↪ winter, Greens
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.title('point cloud segmented')
```

[10]:
```
Text(0.5, 0.92, 'point cloud segmented')
```



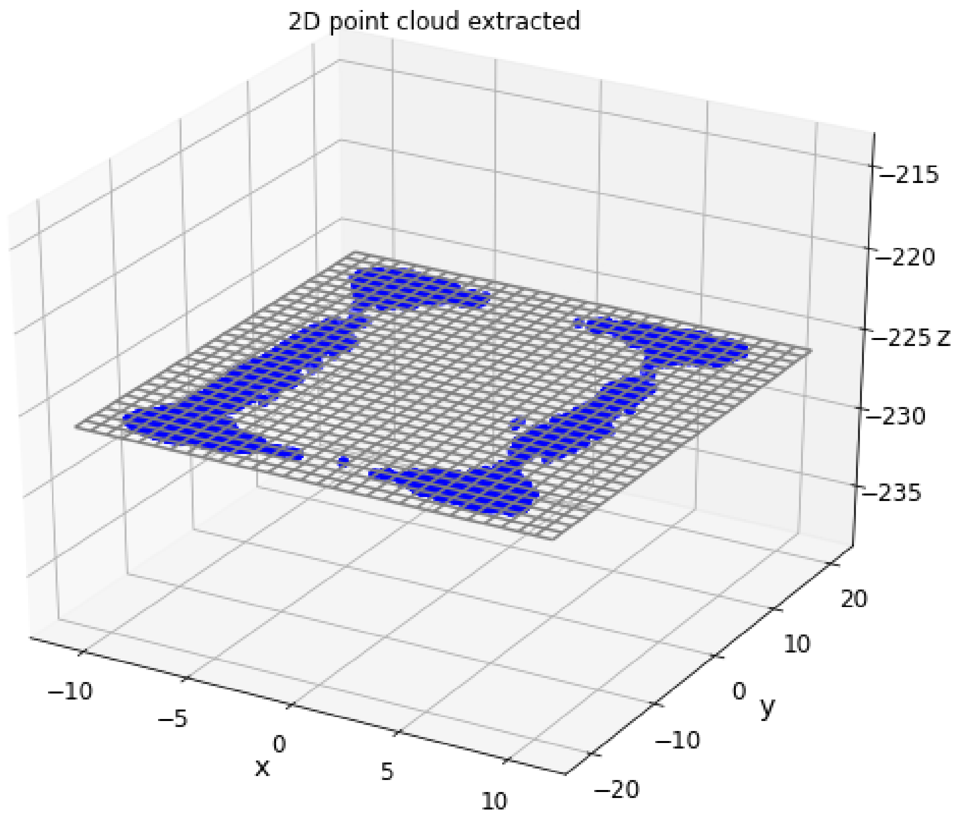
1.6 Extract the 2D projected points from the segmented cloud

```
[11]: ## set the z = z_mean for all the points
      point_extracted = points_segmented
      point_extracted[:,2] = z_mean
      print("the point cloud projected on the 2D plane: ")
      print(point_extracted)
      print ("point_extracted shape: " + str(point_extracted.shape))


      x_2d = point_extracted[:,0]
      y_2d = point_extracted[:,1]
      z_2d = point_extracted[:,2]



      plt.figure(figsize=(10, 8))
      # plt.figure()
```

```
ax = plt.axes(projection='3d')

ax.plot_wireframe(X, Y, Z, color='gray')
# Data for three-dimensional scattered points, extracted from numpy array
ax.scatter3D(x_2d, y_2d, z_2d, c=z_2d, cmap='winter'); # other cmap options:␣
 ↪winter, Greens

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.title('2D point cloud extracted')
```

```
the point cloud projected on the 2D plane:
[[  -9.56302014  -17.74887974 -225.98640157]
 [  -9.56279359  -17.67781856 -225.98640157]
 [  -9.5634848   -17.60777515 -225.98640157]
 …
 [   8.66727035   17.74265523 -225.98640157]
 [   8.66658094   17.81269927 -225.98640157]
 [   8.66716664   17.88435272 -225.98640157]]
point_extracted shape: (17510, 3)
```

```
[11]: Text(0.5, 0.92, '2D point cloud extracted')
```



1.7 Convert all points into 2D points

[12]:
```
print ("point_extracted number of points: " + str(point_extracted.shape[0]))
point_2D_extracted = np.zeros((point_extracted.shape[0], 2))
point_2D_extracted[:, 0] = point_extracted[:, 0]
point_2D_extracted[:, 1] = point_extracted[:, 1]
print ("point_2D_extracted dimensin: " + str(point_2D_extracted.shape))
```

```
point_extracted number of points: 17510
point_2D_extracted dimensin: (17510, 2)
```

1.8 DBSCAN clustering by Open3D

[13]:
```
## produce 2D point cloud with the above numpy array data

# an open3d point cloud object
pcd_2D = o3d.geometry.PointCloud()
```

```
pcd_2D.points = o3d.utility.Vector3dVector(point_extracted)


# with o3d.utility.VerbosityContextManager(o3d.utility.VerbosityLevel.Debug) as␣
↪cm:
labels = np.array(pcd_2D.cluster_dbscan(eps=0.8, min_points=10,␣
↪print_progress=True))

max_label = labels.max()
print(f"point cloud has {max_label + 1} clusters")
colors = plt.get_cmap("tab20")(labels / (max_label if max_label > 0 else 1))
colors[labels < 0] = 0
pcd_2D.colors = o3d.utility.Vector3dVector(colors[:, :3])
o3d.visualization.draw_geometries([pcd_2D])
```

```
point cloud has 5 clusters
```

1.9 check points in each clusters

[14]:
```
print("the clustered point cloud label are: \n")
print(labels)
print("size of the point cloud 2D extracted: " + str(point_2D_extracted.size))
print("size of the lables: " + str(labels.size))
17510 * 2
```

```
the clustered point cloud label are:

[0 0 0 … 2 2 2]
size of the point cloud 2D extracted: 35020
size of the lables: 17510
```

[14]:
```
35020
```

1.10 Downsample the points using voxel grid filter

[15]:
```
print("Downsample the 2D point cloud with a voxel of 0.8")
downsample_pcd_2D = pcd_2D.voxel_down_sample(voxel_size=0.5)

o3d.visualization.draw_geometries([downsample_pcd_2D])
```

```
Downsample the 2D point cloud with a voxel of 0.8
```

Convert the open3D points to numpy array for further processing

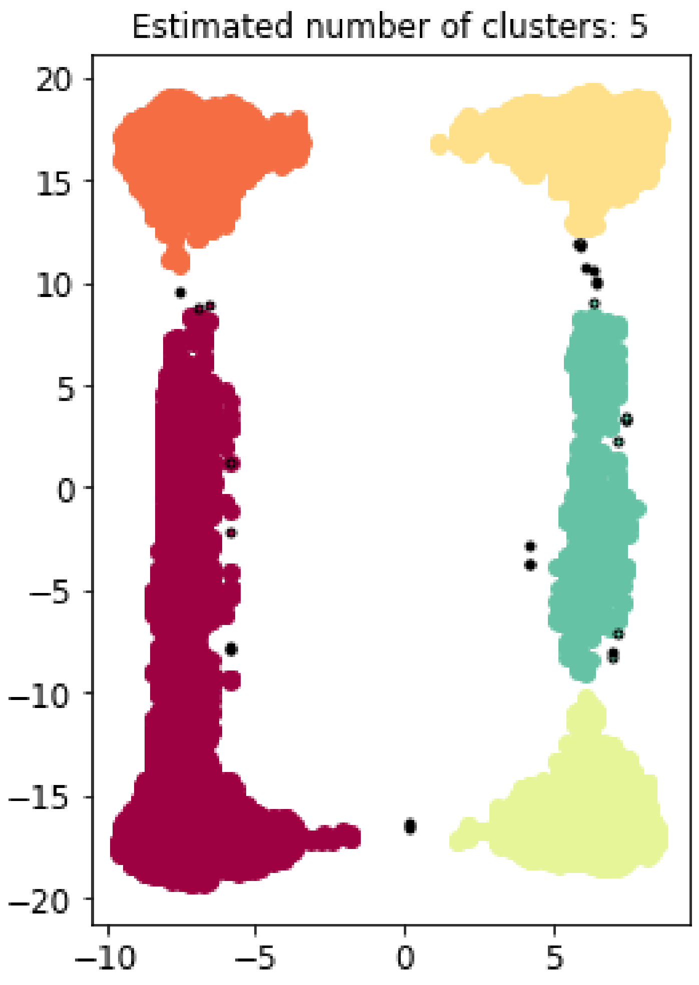
[16]:
```
# convert Open3D.o3d.geometry.PointCloud to numpy array
extracted_2D_downsampled = np.asarray(downsample_pcd_2D.points)
# print (extracted_2D_downsampled)
```

1.11 Using Sklearn DBSCAN clustering

[17]:
```
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler


# #############################################################################
# Compute DBSCAN
db = DBSCAN(eps=0.8, min_samples=10).fit(point_2D_extracted)
# db = DBSCAN(eps=1, min_samples=10).fit(extracted_2D_downsampled)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)

# #############################################################################
# Plot result
import matplotlib.pyplot as plt



plt.figure(figsize=(4, 6))




# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]

# colors = ['r','g','b','c','y']
```

```
point_clusters = []
cluster = 0
while cluster < n_clusters_:

    class_member_mask = (labels == cluster)
    point_clusters.append(point_2D_extracted[class_member_mask &␣
 ↪core_samples_mask])
#     point_clusters.append(extracted_2D_downsampled[class_member_mask &␣
 ↪core_samples_mask])
    cluster = cluster + 1


for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

#     point_clusters[k] =  point_2D_extracted[class_member_mask &␣
 ↪core_samples_mask]

    xy = point_2D_extracted[class_member_mask & core_samples_mask]
#     xy = extracted_2D_downsampled[class_member_mask & core_samples_mask]
#     plt.plot(xy[:, 0], xy[:, 1], '.', markerfacecolor=tuple(col),
#              markeredgecolor='k', markersize=14)
    plt.plot(xy[:, 0], xy[:, 1], '.', markerfacecolor=tuple(col),
             markeredgecolor='None', markersize=14)

    # these are the points does not belong to either clusters (noise)
    xy = point_2D_extracted[class_member_mask & ~core_samples_mask]
#     xy = extracted_2D_downsampled[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], '.', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

```
Estimated number of clusters: 5
Estimated number of noise points: 16
```



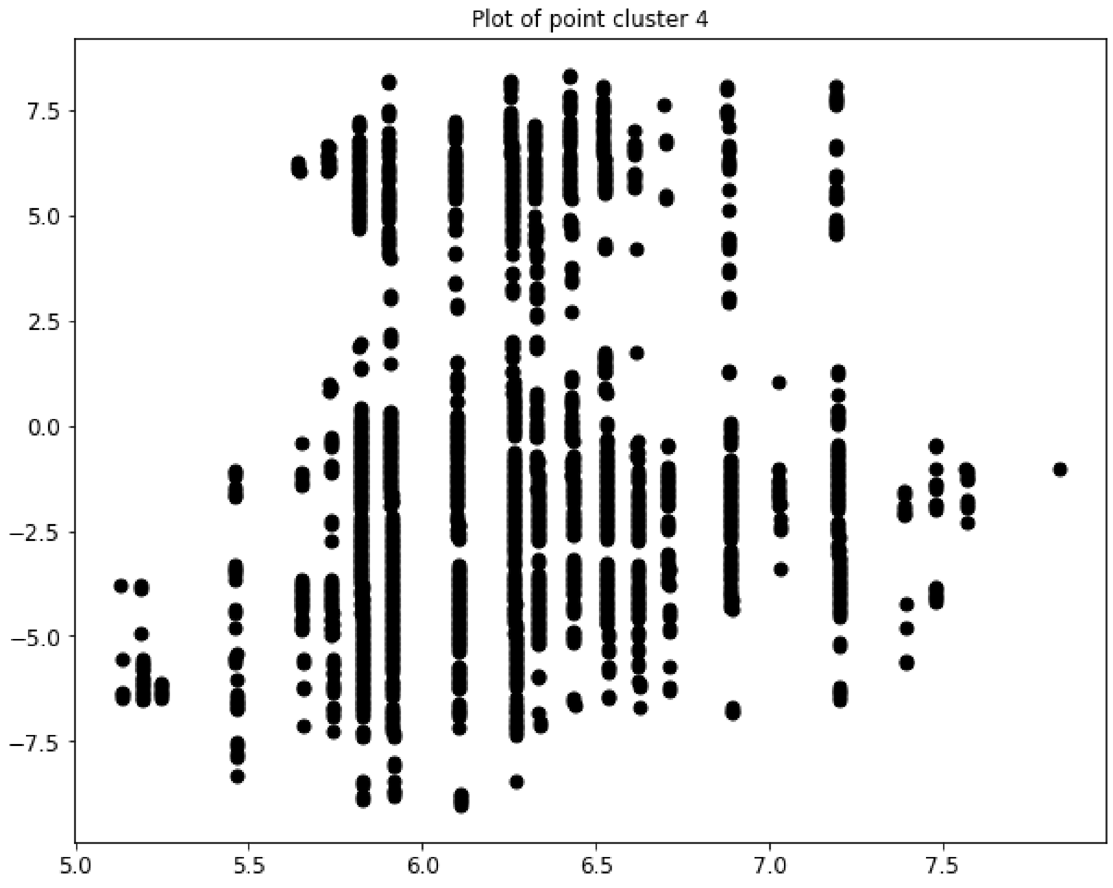
```
[18]: print("the dimension of the point cloud cluster: " + str(point_clusters[4].
       ↪shape))
```

```
the dimension of the point cloud cluster: (1432, 2)
```

1.12 Visualize one indivudual point cluster

```
[19]: ##---- visualize only the point cluster 5
      plt.figure(figsize=(10, 8))


      plt.plot(point_clusters[4][:, 0], point_clusters[4][:, 1], '.',␣
       ↪markerfacecolor=tuple(col),
               markeredgecolor='k', markersize=14)
```

```
plt.title('Plot of point cluster 4')
plt.show()
```



1.13 Get the contour of each point cloud clusters

[20]:
```
# tetra_mesh, pt_map = o3d.geometry.TetraMesh.create_from_point_cloud(pcd)
# # for alpha in np.logspace(np.log10(0.5), np.log10(0.01), num=2):
# for alpha in 0.5,0.4,0.3:
#     print(f"alpha={alpha:.3f}")
#     mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(
#         pcd, alpha, tetra_mesh, pt_map)
#     mesh.compute_vertex_normals()
#     o3d.visualization.draw_geometries([mesh])
```

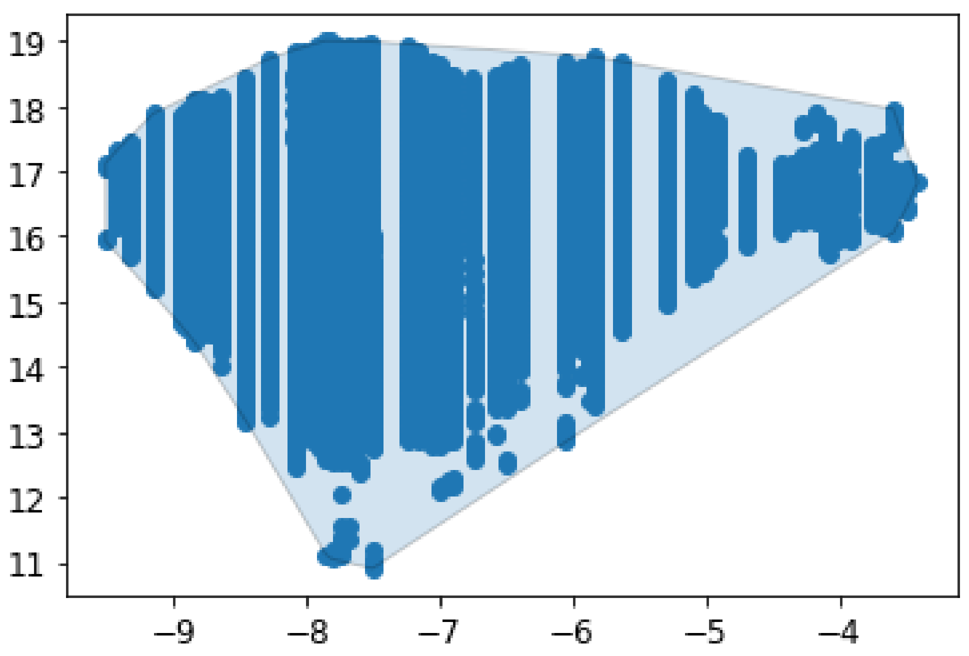
1.13.1 Example: using open3d for concave hull fitting

[21]:
```
## mesh = o3dtut.get_bunny_mesh()

# o3d.visualization.draw_geometries([pcd])
# alpha = 0.03
# print(f"alpha={alpha:.3f}")
# mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(
#     pcd, alpha)
# mesh.compute_vertex_normals()
# o3d.visualization.draw_geometries([mesh])
```

1.13.2 Example : using alphashape python library to extract contour

[22]:
```
import alphashape
import sys
from descartes import PolygonPatch

#-------------1. using self-optimization of alpha␣
 ↪value--------------------------
# alpha = 0.95 * alphashape.optimizealpha(point_clusters[4])
# hull = alphashape.alphashape(point_clusters[4], alpha)
# hull_pts = hull.exterior.coords.xy

# fig, ax = plt.subplots()
# ax.scatter(hull_pts[0], hull_pts[1], color='red')


#-------------2. using user-defined alpha value--------------------------
alpha_shape = alphashape.alphashape(point_clusters[1], 0.1)
plt.figure(figsize=(10, 8))
fig, ax = plt.subplots()
ax.scatter(*zip(*point_clusters[1]))
ax.add_patch(PolygonPatch(alpha_shape, alpha=0.2))
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```

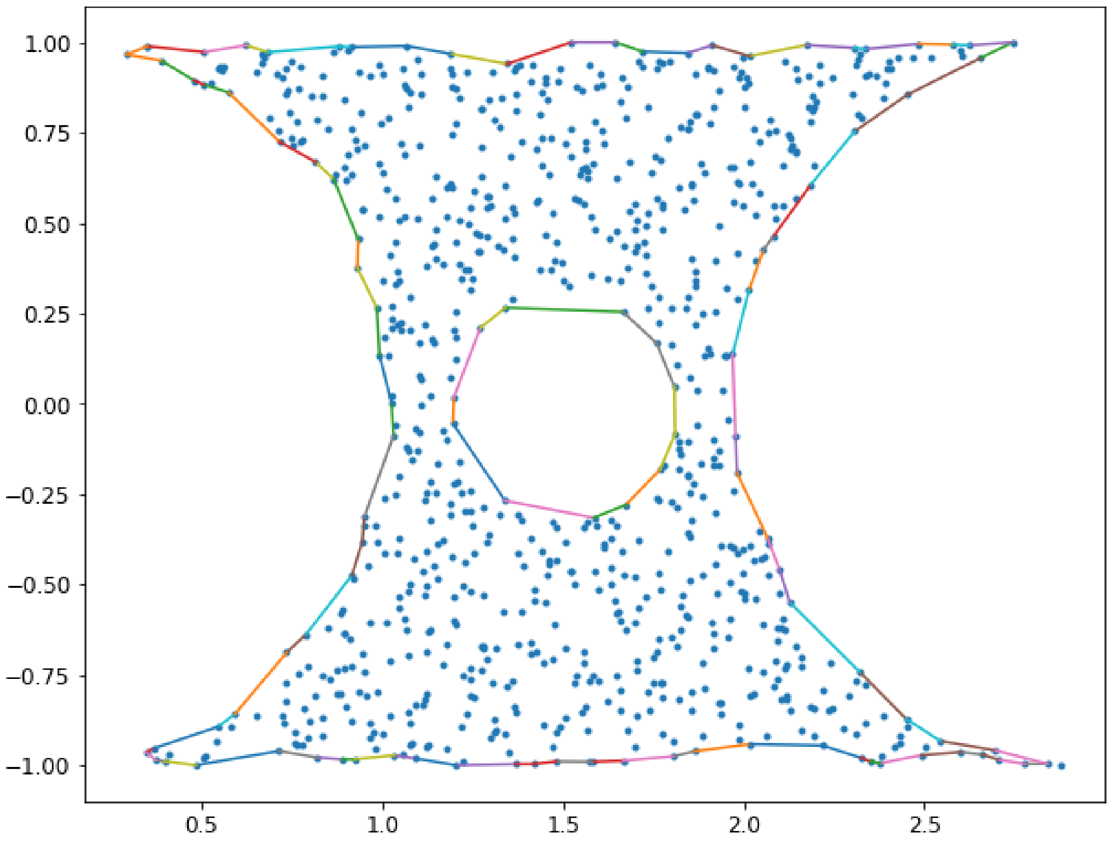


1.13.3 Example : (from online)

https://stackoverflow.com/questions/23073170/calculate-bounding-polygon-of-alpha-shape-from-the-delaunay-triangulation

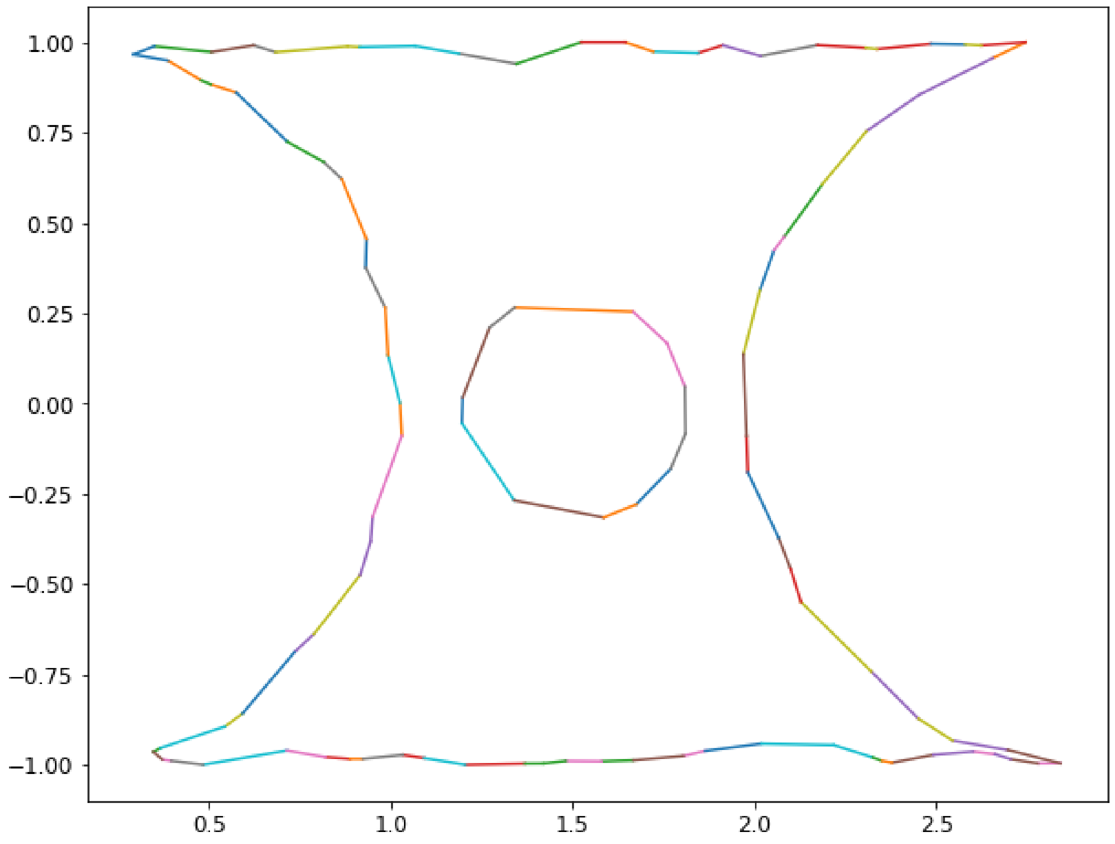
[23]:
```python
from scipy.spatial import Delaunay
import numpy as np


def alpha_shape(points, alpha, only_outer=True):
    """
    Compute the alpha shape (concave hull) of a set of points.
    :param points: np.array of shape (n,2) points.
    :param alpha: alpha value.
    :param only_outer: boolean value to specify if we keep only the outer border
    or also inner edges.
    :return: set of (i,j) pairs representing edges of the alpha-shape. (i,j) are
    the indices in the points array.
    """
    assert points.shape[0] > 3, "Need at least four points"

    def add_edge(edges, i, j):
        """
        Add a line between the i-th and j-th points,
        if not in the list already
        """
        if (i, j) in edges or (j, i) in edges:
            # already added
            assert (j, i) in edges, "Can't go twice over same directed edge␣
↪right?"
            if only_outer:
                # if both neighboring triangles are in shape, it is not a␣
↪boundary edge
                edges.remove((j, i))
            return
        edges.add((i, j))

    tri = Delaunay(points)
    edges = set()
    # Loop over triangles:
    # ia, ib, ic = indices of corner points of the triangle
    for ia, ib, ic in tri.simplices:
        pa = points[ia]
        pb = points[ib]
        pc = points[ic]
        # Computing radius of triangle circumcircle
        # www.mathalino.com/reviewer/derivation-of-formulas/
↪derivation-of-formula-for-radius-of-circumcircle
        a = np.sqrt((pa[0] - pb[0]) ** 2 + (pa[1] - pb[1]) ** 2)
        b = np.sqrt((pb[0] - pc[0]) ** 2 + (pb[1] - pc[1]) ** 2)
        c = np.sqrt((pc[0] - pa[0]) ** 2 + (pc[1] - pa[1]) ** 2)
        s = (a + b + c) / 2.0
        area = np.sqrt(s * (s - a) * (s - b) * (s - c))
        circum_r = a * b * c / (4.0 * area)
        if circum_r < alpha:
            add_edge(edges, ia, ib)
            add_edge(edges, ib, ic)
            add_edge(edges, ic, ia)
    return edges


# from matplotlib.pyplot import *

# Constructing the input point data
np.random.seed(0)
x = 3.0 * np.random.rand(2000)
y = 2.0 * np.random.rand(2000) - 1.0
inside = ((x ** 2 + y ** 2 > 1.0) & ((x - 3) ** 2 + y ** 2 > 1.0) & ((x - 1.5)␣
↪** 2 + y ** 2 > 0.09))
points = np.vstack([x[inside], y[inside]]).T

# Computing the alpha shape
```

```
edges = alpha_shape(points, alpha=0.25, only_outer=True)

# Plotting the output
plt.figure(figsize=(10, 8))
plt.axis('equal')
plt.plot(points[:, 0], points[:, 1], '.')
for i, j in edges:
    plt.plot(points[[i, j], 0], points[[i, j], 1])
plt.show()
```



[24]:
```
print (edges)
# Plotting the output
plt.figure(figsize=(10, 8))

for i, j in edges:
    plt.plot(points[[i, j], 0], points[[i, j], 1])
plt.show()
```

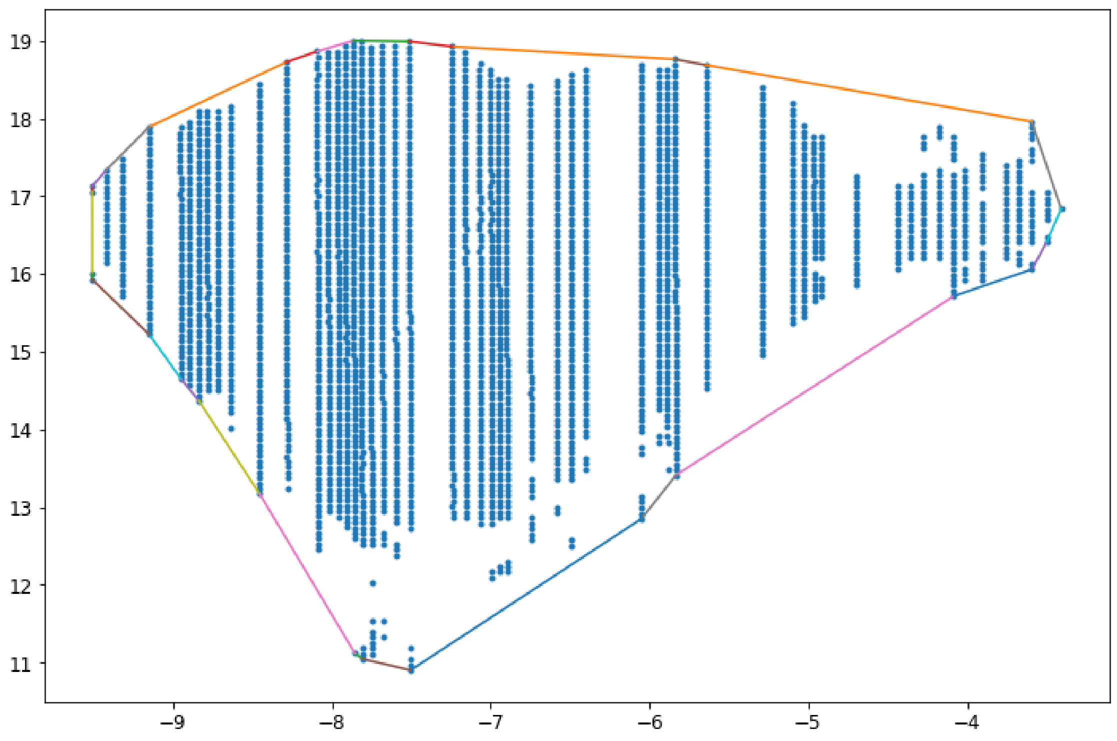
```
{(294, 692), (473, 366), (578, 731), (276, 821), (606, 70), (628, 643), (723,
555), (245, 750), (29, 294), (32, 564), (167, 298), (223, 271), (610, 363),
(787, 13), (483, 336), (409, 723), (658, 80), (630, 183), (829, 766), (26, 743),
(280, 462), (599, 744), (564, 182), (477, 141), (801, 317), (205, 672), (69,
109), (392, 539), (70, 801), (253, 167), (664, 735), (182, 628), (632, 538),
(522, 537), (382, 813), (363, 69), (514, 247), (487, 280), (711, 786), (821,
545), (744, 267), (545, 589), (328, 392), (366, 770), (643, 514), (538, 658),
(39, 1), (109, 26), (634, 483), (114, 829), (289, 477), (183, 664), (267, 630),
(98, 581), (813, 634), (712, 253), (743, 98), (157, 134), (317, 626), (553,
223), (731, 242), (652, 718), (542, 733), (838, 273), (134, 276), (273, 29),
(718, 39), (1, 487), (766, 245), (537, 787), (80, 726), (581, 704), (13, 551),
(626, 205), (681, 636), (750, 578), (741, 632), (735, 263), (733, 681), (726,
32), (672, 838), (154, 165), (165, 599), (573, 157), (247, 409), (298, 3), (692,
542), (3, 652), (770, 289), (539, 114), (336, 711), (263, 553), (551, 741),
(589, 328), (636, 473), (555, 606), (271, 382), (704, 522), (141, 573), (786,
610), (242, 154), (462, 712)}
```



1.13.4 Tuning the parameter for concave hull fitting: alpha

- current value of alpha for fitting: 5 (default)
- for point cloud cluster 1

```
[25]: # Constructing the input point data

      # points = np.vstack([point_clusters[4][:, 0], point_clusters[4][:, 1].T

      # Computing the alpha shape
      edges1 = alpha_shape(point_clusters[1], alpha=5, only_outer=True)


      vertices_x_cluster_1 = []
      vertices_y_cluster_1 = []
      vertices_cluster_edge_test = []

      # Plotting the output
      plt.figure(figsize=(12, 8))
      # plt.axis('equal')
      plt.plot(point_clusters[1][:, 0], point_clusters[1][:, 1], '.')
      for i, j in edges1:
          plt.plot(point_clusters[1][[i, j], 0], point_clusters[1][[i, j], 1])

          vertices_x_cluster_1.append(point_clusters[1][[0,j], 0]) # x coordinates of␣
       ↪polgon vertices, vector (array)
          vertices_y_cluster_1.append( point_clusters[1][[1, j], 1])  # y coordinates␣
       ↪of polgon vertices, vector


      vertices_cluster_edge_test.append (point_clusters[1][[1, 2], 0])
      plt.show()

      print (vertices_x_cluster_1)
      print ("\n")
      print (vertices_cluster_edge_test)
```



```
[array([-9.5099091 , -7.24462729]), array([-9.5099091 , -7.80828165]),
array([-9.5099091 , -9.51050752]), array([-9.5099091 , -8.83804074]),
array([-9.5099091, -5.8371427]), array([-9.5099091, -4.0807922]),
array([-9.5099091 , -5.83194513]), array([-9.5099091 , -9.50976375]),
array([-9.5099091 , -8.95130951]), array([-9.5099091 , -6.04854025]),
array([-9.5099091 , -9.15158739]), array([-9.5099091, -9.5099091]),
array([-9.5099091 , -8.28731867]), array([-9.5099091 , -3.49760769]),
array([-9.5099091 , -7.50422509]), array([-9.5099091 , -8.09237637]),
array([-9.5099091 , -3.59152755]), array([-9.5099091 , -8.45456718]),
array([-9.5099091 , -3.40618803]), array([-9.5099091 , -3.58968265]),
array([-9.5099091 , -5.64080576]), array([-9.5099091 , -7.86364789]),
array([-9.5099091 , -7.51229332]), array([-9.5099091 , -9.51115761]),
array([-9.5099091 , -9.14944738]), array([-9.5099091 , -7.85609261]),
array([-9.5099091 , -9.42201341])]
```

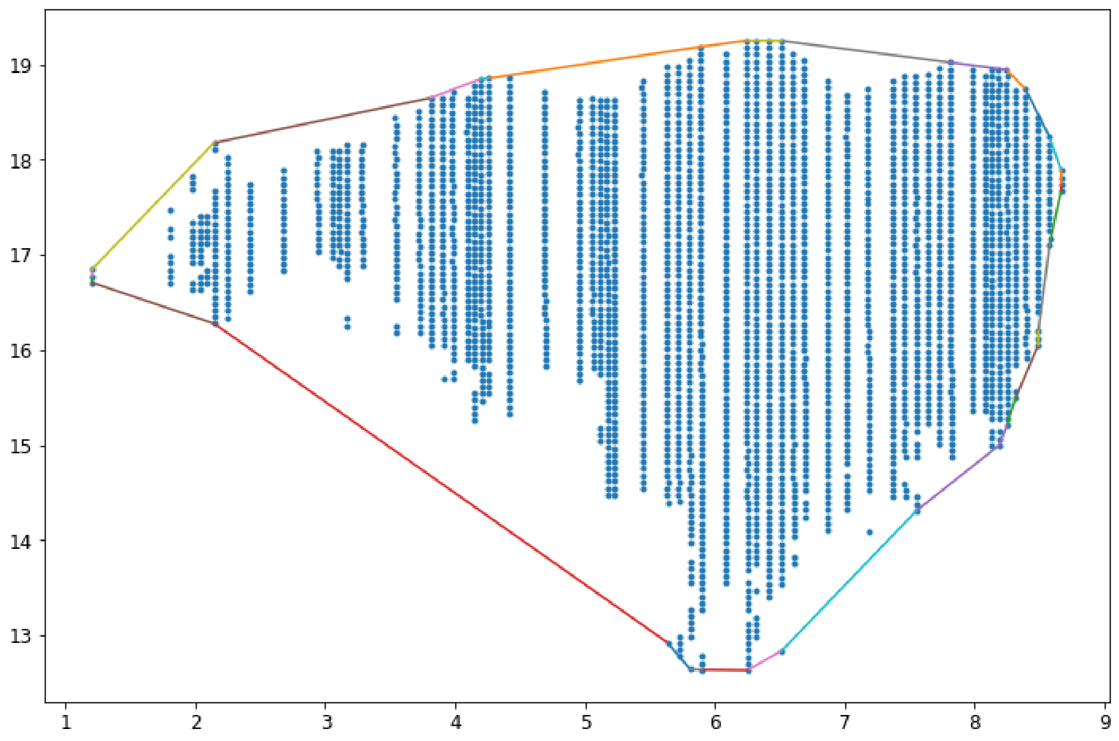
```
[array([-9.50976375, -9.51050752])]
```

[26]:
```
# print("Show the dimension of the edges: " + edges.shape)
print("print the value of edges: " + str(edges1))
```

```
print the value of edges: {(2567, 1555), (905, 998), (3, 2), (87, 183), (2627,
2567), (2491, 2901), (2282, 2491), (2, 1), (48, 87), (1375, 2282), (552, 86),
(1, 0), (644, 552), (3007, 3026), (998, 1375), (997, 644), (3034, 3025), (183,
399), (3026, 3034), (2901, 3007), (3025, 2627), (1468, 997), (1555, 1468), (21,
3), (0, 48), (399, 905), (86, 21)}
```

1.13.5 for point cloud cluster 2 (visualization with polygon fitting)

```
[27]: # Computing the alpha shape
edges2 = alpha_shape(point_clusters[2], alpha=5, only_outer=True)


vertices_x_cluster_2 = []
vertices_y_cluster_2 = []
vertices_cluster_edge_test = []

# Plotting the output
plt.figure(figsize=(12, 8))
# plt.axis('equal')
plt.plot(point_clusters[2][:, 0], point_clusters[2][:, 1], '.')
for i, j in edges2:
    plt.plot(point_clusters[2][[i, j], 0], point_clusters[2][[i, j], 1])

    vertices_x_cluster_2.append(point_clusters[2][[0,j], 0]) # x coordinates of␣
 ↪polgon vertices, vector (array)
    vertices_y_cluster_2.append( point_clusters[2][[1, j], 1])  # y coordinates␣
 ↪of polgon vertices, vector


vertices_cluster_edge_test.append (point_clusters[1][[1, 2], 0])
plt.show()

print (vertices_x_cluster_1)
print ("\n")
print (vertices_cluster_edge_test)
```

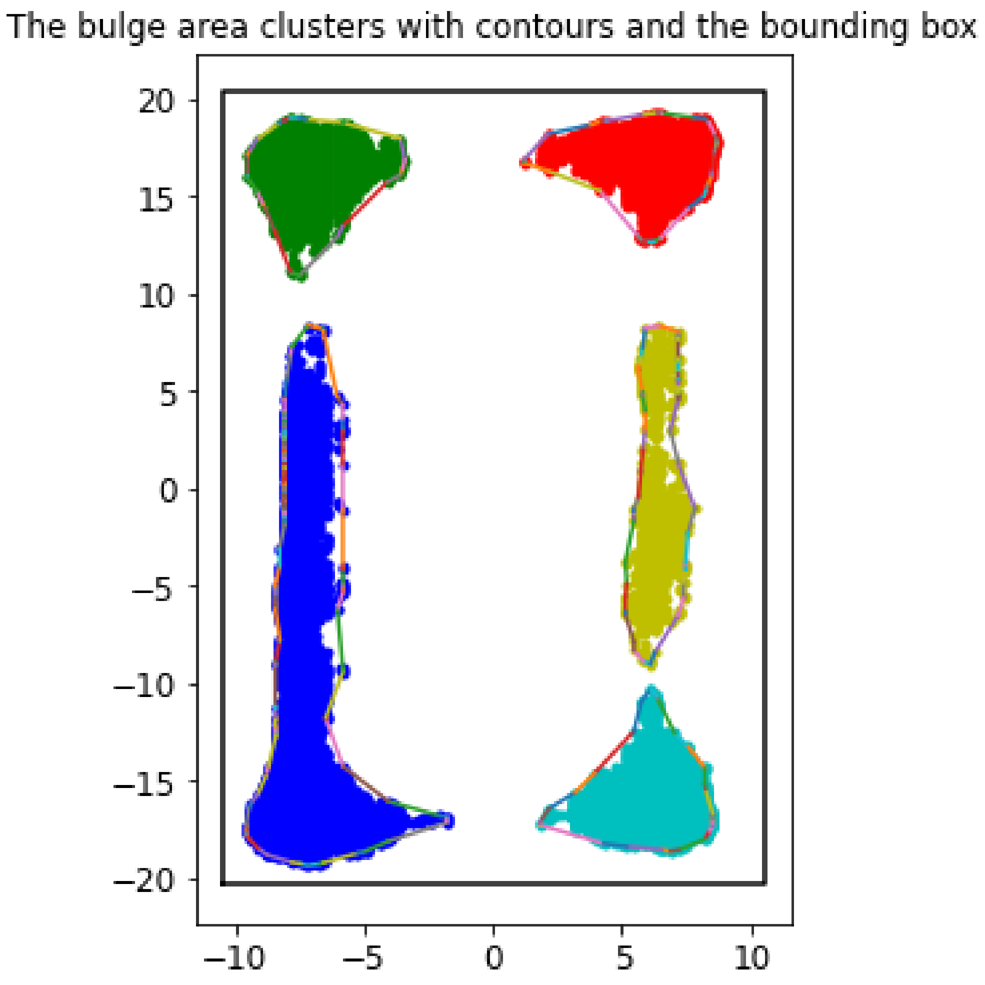


```
[array([-9.5099091 , -7.24462729]), array([-9.5099091 , -7.80828165]),
array([-9.5099091 , -9.51050752]), array([-9.5099091 , -8.83804074]),
array([-9.5099091, -5.8371427]), array([-9.5099091, -4.0807922]),
array([-9.5099091 , -5.83194513]), array([-9.5099091 , -9.50976375]),
array([-9.5099091 , -8.95130951]), array([-9.5099091 , -6.04854025]),
array([-9.5099091 , -9.15158739]), array([-9.5099091, -9.5099091]),
array([-9.5099091 , -8.28731867]), array([-9.5099091 , -3.49760769]),
array([-9.5099091 , -7.50422509]), array([-9.5099091 , -8.09237637]),
array([-9.5099091 , -3.59152755]), array([-9.5099091 , -8.45456718]),
array([-9.5099091 , -3.40618803]), array([-9.5099091 , -3.58968265]),
array([-9.5099091 , -5.64080576]), array([-9.5099091 , -7.86364789]),
array([-9.5099091 , -7.51229332]), array([-9.5099091 , -9.51115761]),
array([-9.5099091 , -9.14944738]), array([-9.5099091 , -7.85609261]),
array([-9.5099091 , -9.42201341])]
```

```
[array([-9.50976375, -9.51050752])]
```

1.14 Plot the point cloud clusters and the bounding box (reference plane)

[28]:
```python
# Computing the alpha shape
edges0 = alpha_shape(point_clusters[0], alpha=4, only_outer=True)
edges2 = alpha_shape(point_clusters[2], alpha=4, only_outer=True)
edges3 = alpha_shape(point_clusters[3], alpha=4, only_outer=True)
edges4 = alpha_shape(point_clusters[4], alpha=4, only_outer=True)

# Plotting the output
plt.figure(figsize=(4, 6))
# plt.axis('equal')
plt.plot(point_clusters[0][:, 0], point_clusters[0][:, 1], 'b.')
plt.plot(point_clusters[1][:, 0], point_clusters[1][:, 1], 'g.')
plt.plot(point_clusters[2][:, 0], point_clusters[2][:, 1], 'r.')
plt.plot(point_clusters[3][:, 0], point_clusters[3][:, 1], 'c.')
plt.plot(point_clusters[4][:, 0], point_clusters[4][:, 1], 'y.')
for i, j in edges0:
    plt.plot(point_clusters[0][[i, j], 0], point_clusters[0][[i, j], 1])
for i, j in edges1:
    plt.plot(point_clusters[1][[i, j], 0], point_clusters[1][[i, j], 1])
for i, j in edges2:
    plt.plot(point_clusters[2][[i, j], 0], point_clusters[2][[i, j], 1])
for i, j in edges3:
    plt.plot(point_clusters[3][[i, j], 0], point_clusters[3][[i, j], 1])
for i, j in edges4:
    plt.plot(point_clusters[4][[i, j], 0], point_clusters[4][[i, j], 1])

# Outer boundary
xd=[-10.5, 10.5, 10.5,-10.5,-10.5];
yd=[-20.3,-20.3, 20.3, 20.3,-20.3];

plt.plot(xd,yd,'k');
plt.title("The bulge area clusters with contours and the bounding box")

plt.show()
```



1.15 Another method for Concave hull (polygon fitting)

—- This one will be chosen

1.15.1 1. ConcaveHull.py from https://gist.github.com/AndreLester/589ea1eddd3a28d00f3d7e47b

http://www.rotefabrik.free.fr/concave_hull/

Advantage: fast, directly give the boundaries of edges with points in numpy array format

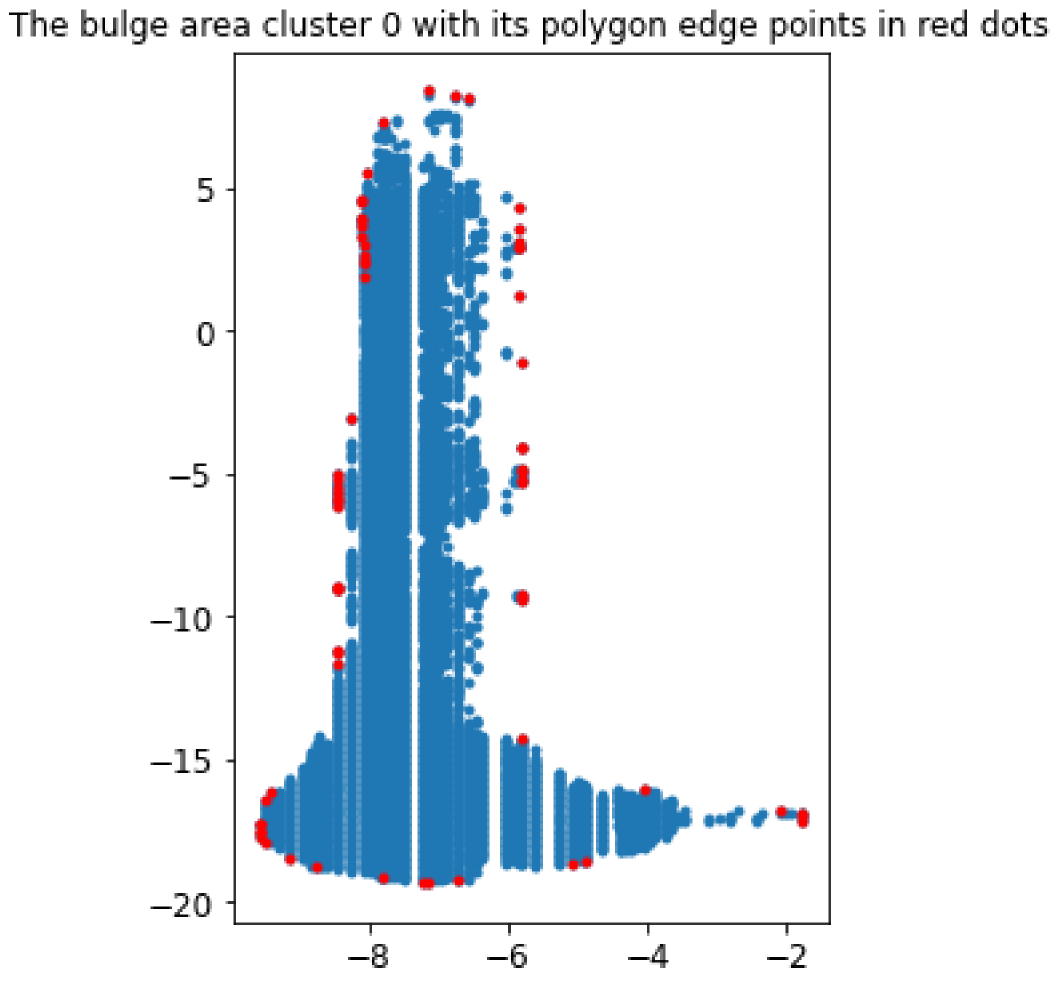
[29]:
```
'''
Copyright (C) 2018  Andre Lester Kruger

ConcaveHull.py is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
```

```
the Free Software Foundation, either version 2 of the License, or
(at your option) any later version.

ConcaveHull.py is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with ConcaveHull.py.  If not, see <http://www.gnu.org/licenses/>.
'''

import bisect
from collections import OrderedDict
import math
#import numpy as np
import matplotlib.tri as tri
from shapely.geometry import LineString
from shapely.geometry import Polygon
from shapely.ops import linemerge


class ConcaveHull:

    def __init__(self):
        self.triangles = {}
        self.crs = {}


    def loadpoints(self, points):
        #self.points = np.array(points)
        self.points = points


    def edge(self, key, triangle):
        '''Calculate the length of the triangle's outside edge
        and returns the [length, key]'''
        pos = triangle[1].index(-1)
        if pos==0:
            x1, y1 = self.points[triangle[0][0]]
            x2, y2 = self.points[triangle[0][1]]
        elif pos==1:
            x1, y1 = self.points[triangle[0][1]]
            x2, y2 = self.points[triangle[0][2]]
        elif pos==2:
            x1, y1 = self.points[triangle[0][0]]
            x2, y2 = self.points[triangle[0][2]]
        length = ((x1-x2)**2+(y1-y2)**2)**0.5
        rec = [length, key]
        return rec


    def triangulate(self):

        if len(self.points) < 2:
            raise Exception('CountError: You need at least 3 points to␣
↪Triangulate')

        temp = list(zip(*self.points))
        x, y = list(temp[0]), list(temp[1])
        del(temp)

        triang = tri.Triangulation(x, y)

        self.triangles = {}

        for i, triangle in enumerate(triang.triangles):
            self.triangles[i] = [list(triangle), list(triang.neighbors[i])]


    def calculatehull(self, tol=50):

        self.tol = tol

        if len(self.triangles) == 0:
            self.triangulate()

        # All triangles with one boundary longer than the tolerance (self.tol)
        # is added to a sorted deletion list.
        # The list is kept sorted from according to the boundary edge's length
        # using bisect
        deletion = []
        self.boundary_vertices = set()
        for i, triangle in self.triangles.items():
            if -1 in triangle[1]:
                for pos, neigh in enumerate(triangle[1]):
                    if neigh == -1:
                        if pos == 0:
                            self.boundary_vertices.add(triangle[0][0])
                            self.boundary_vertices.add(triangle[0][1])
                        elif pos == 1:
                            self.boundary_vertices.add(triangle[0][1])
                            self.boundary_vertices.add(triangle[0][2])
                        elif pos == 2:
                            self.boundary_vertices.add(triangle[0][0])
                            self.boundary_vertices.add(triangle[0][2])
            if -1 in triangle[1] and triangle[1].count(-1) == 1:
                rec = self.edge(i, triangle)
                if rec[0] > self.tol and triangle[1].count(-1) == 1:
                    bisect.insort(deletion, rec)

        while len(deletion) != 0:
            # The triangles with the longest boundary edges will be
            # deleted first
            item = deletion.pop()
            ref = item[1]
            flag = 0

            # Triangle will not be deleted if it already has two boundary edges␣
↪

            if self.triangles[ref][1].count(-1) > 1:
                continue

            # Triangle will not be deleted if the inside node which is not
            # on this triangle's boundary is already on the boundary of
            # another triangle
            adjust = {0: 2, 1: 0, 2: 1}
            for i, neigh in enumerate(self.triangles[ref][1]):
                j = adjust[i]
                if neigh == -1 and self.triangles[ref][0][j] in self.
↪boundary_vertices:
                    flag = 1
                    break
            if flag == 1:
                continue

            for i, neigh in enumerate(self.triangles[ref][1]):
                if neigh == -1:
                    continue
                pos = self.triangles[neigh][1].index(ref)
                self.triangles[neigh][1][pos] = -1
                rec = self.edge(neigh, self.triangles[neigh])
                if rec[0] > self.tol and self.triangles[rec[1]][1].count(-1) ==␣
↪1:
                    bisect.insort(deletion, rec)

            for pt in self.triangles[ref][0]:
                self.boundary_vertices.add(pt)

            del self.triangles[ref]
        self.polygon()



    def polygon(self):

        edgelines = []
        for i, triangle in self.triangles.items():
            if -1 in triangle[1]:
                for pos, value in enumerate(triangle[1]):
                    if value == -1:
                        if pos==0:
                            x1, y1 = self.points[triangle[0][0]]
                            x2, y2 = self.points[triangle[0][1]]
                        elif pos==1:
                            x1, y1 = self.points[triangle[0][1]]
                            x2, y2 = self.points[triangle[0][2]]
                        elif pos==2:
                            x1, y1 = self.points[triangle[0][0]]
                            x2, y2 = self.points[triangle[0][2]]
                        line = LineString([(x1, y1), (x2, y2)])
                        edgelines.append(line)

        bound = linemerge(edgelines)

        self.boundary = Polygon(bound.coords)



#if __name__ == '__main__':
```

1.15.2 visualization (cluster 0)

- Hyperparameter for tuning: tol (equivalent to alpha value)

[30]:
```
# from ConcaveHull import ConcaveHull

ch = ConcaveHull()
# pts = np.random.uniform(size=(100, 2))
pts = point_clusters[0]
ch.loadpoints(pts)
ch.calculatehull(tol = 5)      ## a hyperparameter for tuning

boundary_points = np.vstack(ch.boundary.exterior.coords.xy).T
# boundary_points is a subset of pts corresponding to the concave hull
# Computing the alpha shape
# Plotting the output
plt.figure(figsize=(4, 6))
plt.plot(point_clusters[0][:, 0], point_clusters[0][:, 1], '.')
plt.plot(boundary_points[:, 0], boundary_points[:, 1], 'r.')


plt.title("The bulge area cluster 0 with its polygon edge points in red dots")

plt.show()

print (boundary_points)
print (boundary_points.shape)
```



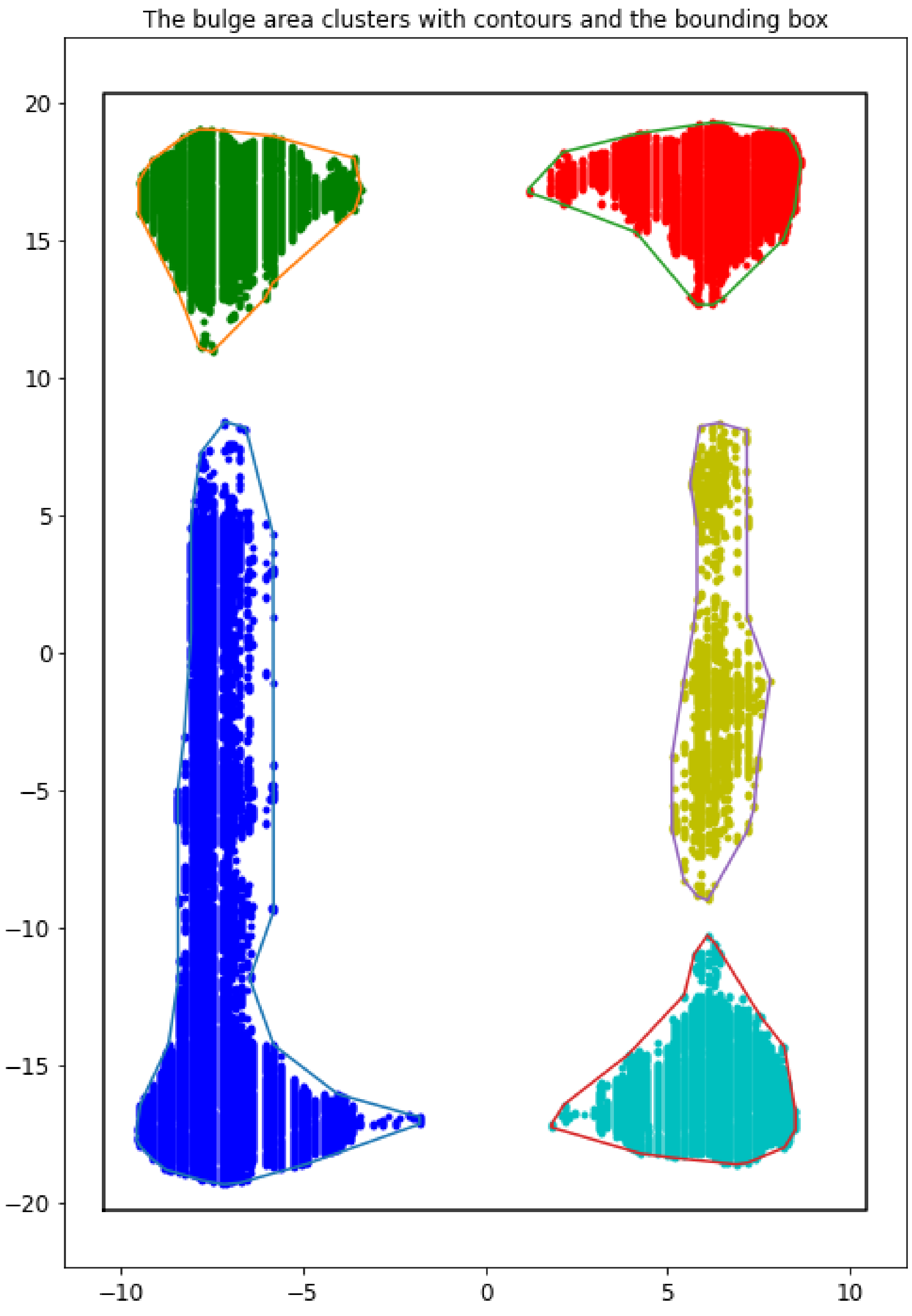
```
[[ -9.5634848  -17.60777515]
 [ -9.56302014 -17.74887974]
 [ -9.47851699 -17.95959273]
 [ -9.11761151 -18.46083308]
 [ -8.75113822 -18.81763292]
 [ -7.77992328 -19.19316765]
 [ -7.20933058 -19.33870567]
 [ -7.12406684 -19.34029991]
 [ -6.71589772 -19.27917588]
 [ -5.06245858 -18.66787476]
 [ -4.88287537 -18.5976883 ]
 [ -1.79843862 -17.15817805]
 [ -1.79878351 -17.01800267]
 [ -1.799247   -16.87584928]
 [ -2.08671842 -16.79918684]
 [ -4.05136342 -16.0624997 ]
 [ -5.80649212 -14.28080698]
 [ -5.81072562  -9.42281761]
 [ -5.81097803  -9.35038047]
 [ -5.81098954  -9.28051593]
 [ -5.81490425  -5.34086754]
 [ -5.81453252  -5.20086149]
 [ -5.81524029  -4.98866294]
 [ -5.81518631  -4.84785563]
 [ -5.8162692   -4.14618928]
 [ -5.81552418  -4.07646385]
 [ -5.81907176  -1.12638768]
 [ -5.8212472    1.25960566]
 [ -5.82288625   2.87364236]
 [ -5.82280902   2.94393488]
 [ -5.82275968   3.08402917]
 [ -5.82332879   3.57611922]
 [ -5.82416619   4.2779746 ]
 [ -6.57583444   8.15191707]
 [ -6.74090133   8.22344095]
 [ -7.15063062   8.37216051]
 [ -7.80542342   7.2531583 ]
 [ -8.0109103    5.50378058]
 [ -8.07879649   4.59424184]
 [ -8.07918652   4.52408035]
 [ -8.07823117   3.89255718]
 [ -8.07829742   3.82058648]
 [ -8.07846502   3.61152406]
 [ -8.07812613   3.26002296]
 [ -8.07779799   2.97939518]
 [ -8.07729263   2.62974456]
 [ -8.07737338   2.41804437]
 [ -8.07724337   2.34757565]
 [ -8.0767623    1.85814305]
 [ -8.26640207  -3.05505461]
```

```
 [ -8.43766541  -5.01836981]
 [ -8.43771175  -5.22867673]
 [ -8.43749905  -5.51074743]
 [ -8.43722327  -5.72083669]
 [ -8.43696015  -5.8631184 ]
 [ -8.43677576  -6.00240427]
 [ -8.43675178  -6.14280669]
 [ -8.43414444  -8.95658272]
 [ -8.43421475  -9.0264872 ]
 [ -8.43376383  -9.09672888]
 [ -8.43224339 -11.21056001]
 [ -8.43143144 -11.28053469]
 [ -8.43123926 -11.70301736]
 [ -9.3913209  -16.19831782]
 [ -9.47948798 -16.47694119]
 [ -9.56294605 -17.25528595]
 [ -9.56305055 -17.32317266]
 [ -9.5634848  -17.60777515]]
(68, 2)
```

1.15.3 plot the whole picture edges (polygon boundary) with point cloud clusters and bounding box

[31]:
```
# from ConcaveHull import ConcaveHull

ch0 = ConcaveHull()
ch1 = ConcaveHull()
ch2 = ConcaveHull()
ch3 = ConcaveHull()
ch4 = ConcaveHull()

pts0 = point_clusters[0]
pts1 = point_clusters[1]
pts2 = point_clusters[2]
pts3 = point_clusters[3]
pts4 = point_clusters[4]

ch0.loadpoints(pts0)
ch1.loadpoints(pts1)
ch2.loadpoints(pts2)
ch3.loadpoints(pts3)
ch4.loadpoints(pts4)

ch0.calculatehull(tol = 4)     ## a hyperparameter for tuning
ch1.calculatehull(tol = 4)
ch2.calculatehull(tol = 4)
```

```
ch3.calculatehull(tol = 4)
ch4.calculatehull(tol = 4)

boundary_points0 = np.vstack(ch0.boundary.exterior.coords.xy).T
boundary_points1 = np.vstack(ch1.boundary.exterior.coords.xy).T
boundary_points2 = np.vstack(ch2.boundary.exterior.coords.xy).T
boundary_points3 = np.vstack(ch3.boundary.exterior.coords.xy).T
boundary_points4 = np.vstack(ch4.boundary.exterior.coords.xy).T
# boundary_points is a subset of pts corresponding to the concave hull

# Plotting the output
plt.figure(figsize=(8, 12))
# plt.axis('equal')
plt.plot(point_clusters[0][:, 0], point_clusters[0][:, 1], 'b.')
plt.plot(point_clusters[1][:, 0], point_clusters[1][:, 1], 'g.')
plt.plot(point_clusters[2][:, 0], point_clusters[2][:, 1], 'r.')
plt.plot(point_clusters[3][:, 0], point_clusters[3][:, 1], 'c.')
plt.plot(point_clusters[4][:, 0], point_clusters[4][:, 1], 'y.')

plt.plot(boundary_points0[:, 0], boundary_points0[:, 1])
plt.plot(boundary_points1[:, 0], boundary_points1[:, 1])
plt.plot(boundary_points2[:, 0], boundary_points2[:, 1])
plt.plot(boundary_points3[:, 0], boundary_points3[:, 1])
plt.plot(boundary_points4[:, 0], boundary_points4[:, 1])

# Outer boundary
xd=[-10.5, 10.5, 10.5,-10.5,-10.5]
yd=[-20.3,-20.3, 20.3, 20.3,-20.3]

plt.plot(xd,yd,'k');
plt.title("The bulge area clusters with contours and the bounding box")

plt.show()
```


The bulge area clusters with contours and the bounding box
20
15
10
5
0
−5
−10
−15
−20
−10
−5
0
5
10

2 Tool path generation

2.1 polyline.py from OpenLMD

2.2 mlabplot.py from OpenLMD

2.3 Contours.py from OpenLMD

3 Polygon infill path generate

https://stackoverflow.com/questions/61853250/polygon-infill-path-generate

https://stackoverflow.com/questions/15668149/polygon-infill-algorithm

https://github.com/Tannz0rz/Mandoline

https://www.mathworks.com/matlabcentral/answers/158900-plotting-zigzag-in-a-2d-contour

https://github.com/nahyunkwon/gcode-modification/blob/87e896818704765201d1f3bcf8604f44d23338fa/adhesion

4 1. Python inpolygon function

- Need to find a suitable function performing "inpolygon" equivalent as matlab
- Matlab inpolygons function:
  - It allows the input polygon vertices to describe multiple NaN-delimited polygons.
  - The polygons can also include holes.

  - returns true/ false to tell if the points are lie in the polygon

4.1 1.1 Method one - matplotlib import path

https://stackoverflow.com/questions/31542843/inpolygon-for-python-examples-of-matplotlib-path-path-contains-points-method

[32]:
```
from matplotlib import path
p = path.Path([(0,0), (0, 1), (1, 1), (1, 0)])  # square with legs length 1 and␣
 ↪bottom left corner at the origin (CW)
p.contains_points([(.5, .5)])
```

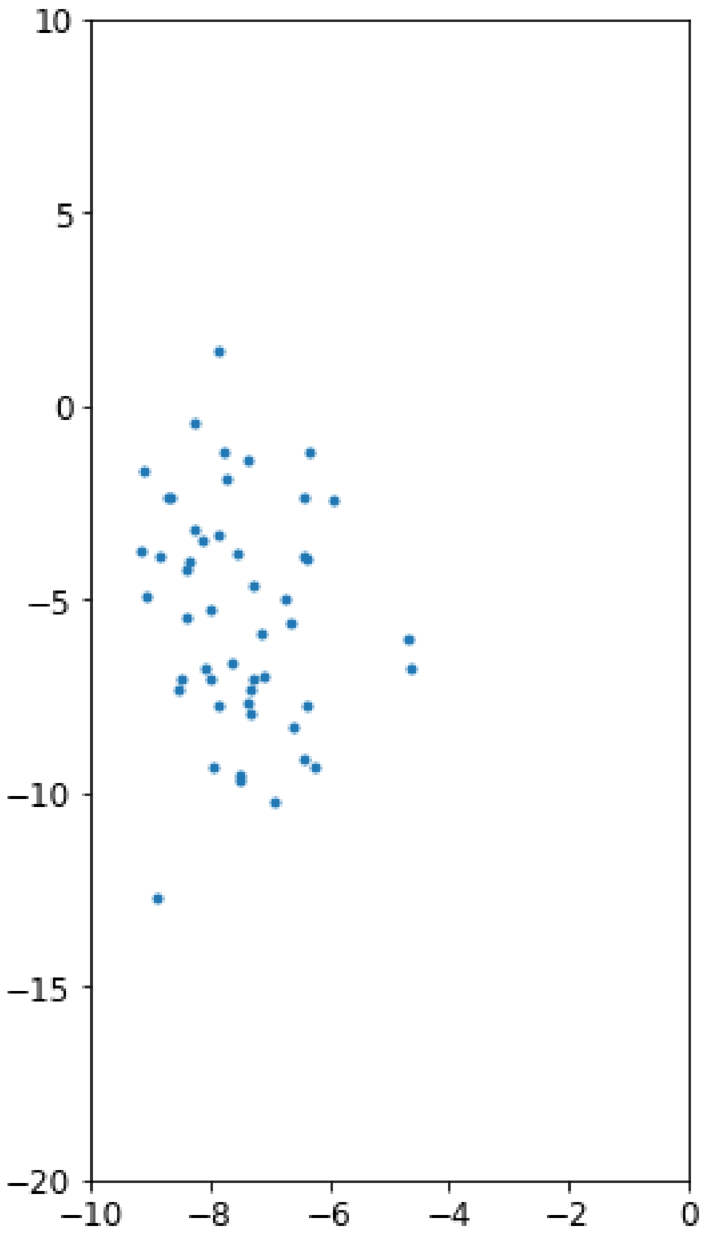
[32]:
```
array([ True])
```

[33]:
```
# use a numpy array of points as well:
points = np.array([.5, .5]).reshape(1, 2)
# >>> points
# array([[ 0.5,  0.5]])
p.contains_points(points)
```

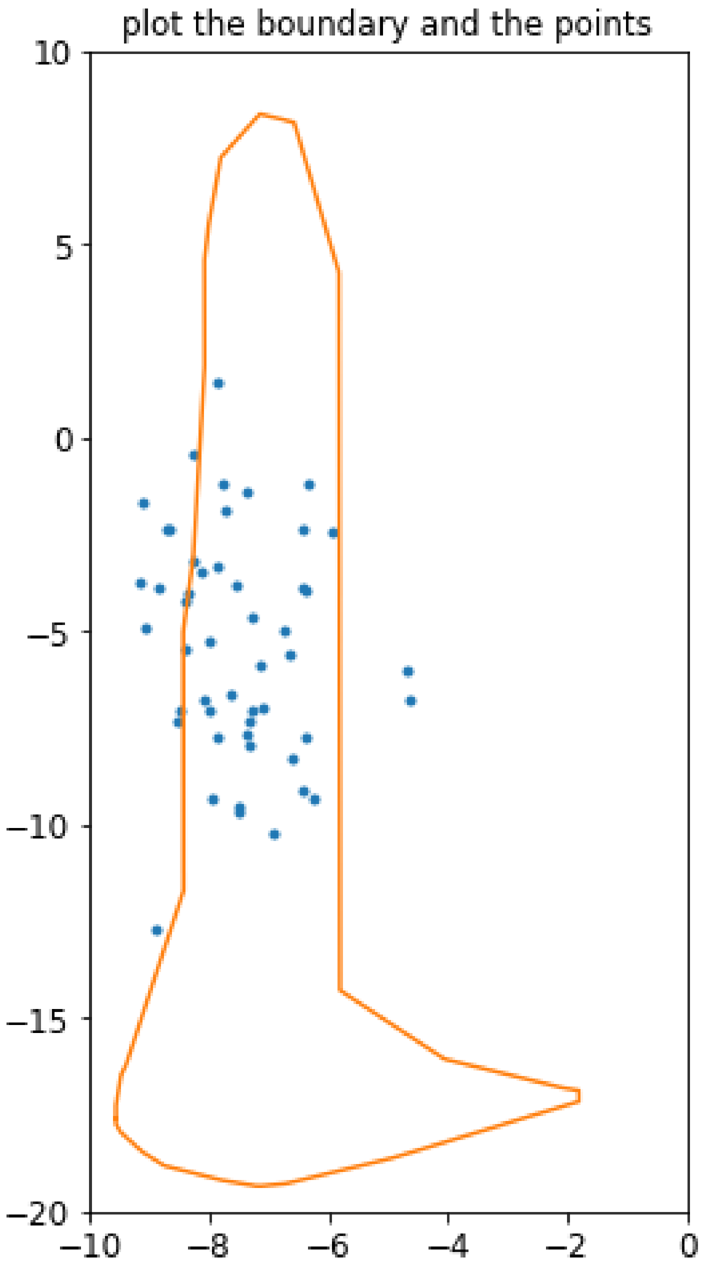
```
[33]: array([ True])
```

Example

```
[34]: ## generate some random points
      # np.random.seed(1)
      xy_points_random = np.random.randn(50,2) ; # dimension 250 x 2
      xy_points_random[:,0] = xy_points_random[:,0] - 7.5
      xy_points_random[:,1] = xy_points_random[:,1] * 3 - 5

      plt.figure(figsize=(4, 8))
      plt.plot(xy_points_random[:, 0], xy_points_random[:, 1], '.')
      plt.xlim([-10, 0])
      plt.ylim([-20, 10])
      plt.show()
```



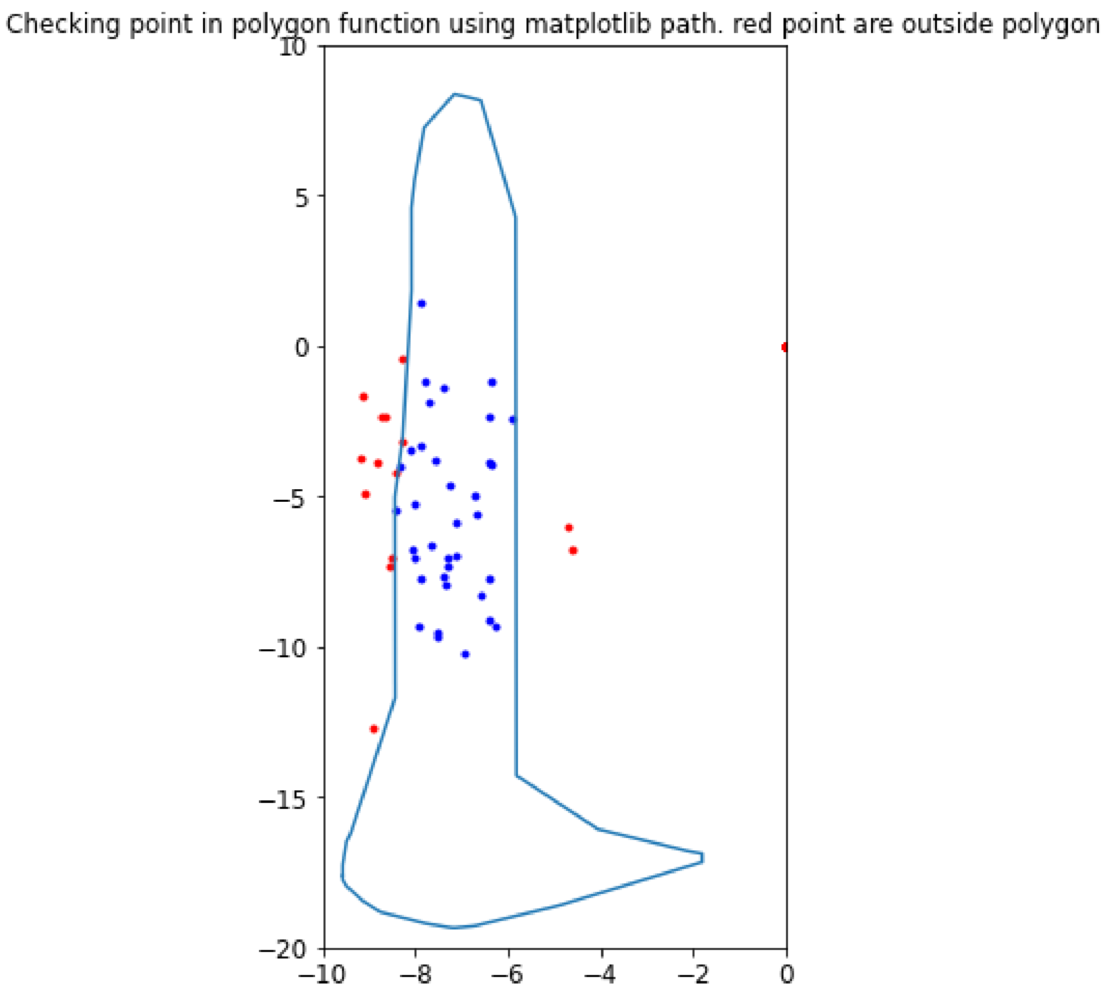
[35]:
```
# create a polygon path with 'boundary_points' generated before (one of the␣
↪hole)
p = path.Path(boundary_points)

plt.figure(figsize=(4, 8))

plt.plot(xy_points_random[:, 0], xy_points_random[:, 1], '.')
plt.plot(boundary_points[:, 0], boundary_points[:, 1])
plt.xlim([-10, 0])
plt.ylim([-20, 10])
plt.title("plot the boundary and the points")
plt.show()
```



[36]:
```
## Determine whether each point lies inside or on the edge of the polygon area.
# point_in = []
point_in = p.contains_points(xy_points_random)
print (point_in)
print (point_in.shape)
```

```
[ True  True  True  True  True  True  True  True  True  True False  True
  True False False  True False False  True  True  True False False  True
  True  True False  True  True False False  True  True  True  True  True
  True  True False  True  True  True  True  True False False  True  True
 False  True]
(50,)
```

[37]:
```
## visualize the graph
# Plotting the output
plt.figure(figsize=(4, 8))

plt.plot(xy_points_random[:, 0]*point_in, xy_points_random[:, 1]*point_in, 'b.')
plt.plot(xy_points_random[:, 0]*(~point_in), xy_points_random[:,␣
 ↪1]*(~point_in), 'r.')

plt.plot(boundary_points[:, 0], boundary_points[:, 1])

plt.xlim([-10, 0])
plt.ylim([-20, 10])
plt.title("Checking point in polygon function using matplotlib path. red point␣
 ↪are outside polygon")
plt.show()
```



4.2 Convert this method to the matlab inpolygon equivalent

4.2.1 Disadvantage:

- cannot return if the point is on the edge of the polygon
- difficult to use if polygon contains holes

[38]:
```
def inpolygon(xq, yq, xv, yv):
    shape = xq.shape
    xq = xq.reshape(-1)
    yq = yq.reshape(-1)
    xv = xv.reshape(-1)
    yv = yv.reshape(-1)
    q = [(xq[i], yq[i]) for i in range(xq.shape[0])]
    p = path.Path([(xv[i], yv[i]) for i in range(xv.shape[0])])
    return p.contains_points(q).reshape(shape)
```

[39]:
```
xv = np.array([0.5,0.2,1.0,0,0.8,0.5])
yv = np.array([1.0,0.1,0.7,0.7,0.1,1])
xq = np.array([0.1,0.5,0.9,0.2,0.4,0.5,0.5,0.9,0.6,0.8,0.7,0.2])
yq = np.array([0.4,0.6,0.9,0.7,0.3,0.8,0.2,0.4,0.4,0.6,0.2,0.6])
print(inpolygon(xq, yq, xv, yv))
```

```
[False False False  True  True  True False False False  True  True  True]
```

4.3 Method 2 - Shapely

https://gis.stackexchange.com/questions/170264/python-point-in-polygon-boundary-and-vertex-check-ray-casting

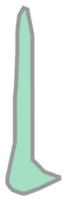
https://shapely.readthedocs.io/en/latest/manual.html#binary-predicates

[40]:
```
from shapely.geometry import Point, Polygon,LinearRing, MultiPoint

## create a polygon object using 'boundary_points'
polygon = Polygon(boundary_points)
```

[41]:
```
# boundary of the polygon = LinearRing
linearring = LinearRing(list(polygon.exterior.coords))
print (linearring)
polygon
```

```
LINEARRING (-9.563484803 -17.60777515, -9.563020136 -17.74887974,
-9.478516986000001 -17.95959273, -9.117611514 -18.46083308, -8.751138219
-18.81763292, -7.779923279 -19.19316765, -7.20933058 -19.33870567, -7.124066844
-19.34029991, -6.715897721 -19.27917588, -5.062458582 -18.66787476, -4.882875368
-18.5976883, -1.798438623 -17.15817805, -1.798783509 -17.01800267, -1.799246999
-16.87584928, -2.086718422 -16.79918684, -4.051363424 -16.0624997, -5.80649212
-14.28080698, -5.810725621 -9.422817607000001, -5.81097803 -9.350380465000001,
-5.810989539 -9.280515926, -5.814904248 -5.340867539, -5.814532519 -5.200861489,
-5.815240292 -4.988662938, -5.815186309 -4.847855625, -5.816269199 -4.146189281,
-5.815524175 -4.076463852, -5.819071757 -1.126387678, -5.821247198 1.259605659,
-5.822886251 2.873642357, -5.822809018 2.943934876, -5.822759678 3.084029174,
-5.823328785 3.576119223, -5.824166189 4.277974596, -6.575834436
8.151917065999999, -6.740901329 8.223440951000001, -7.150630623
8.372160513000001, -7.805423417 7.253158296, -8.010910299000001 5.503780578,
-8.078796486 4.594241844, -8.079186519 4.524080346, -8.078231165 3.892557185,
-8.078297421 3.820586477, -8.078465016999999 3.61152406, -8.078126127000001
3.260022964, -8.077797985 2.979395178, -8.077292632000001 2.62974456,
-8.077373380999999 2.418044374, -8.077243371 2.347575652, -8.076762302000001
1.858143052, -8.266402065999999 -3.055054607, -8.437665414 -5.018369815,
-8.437711748 -5.22867673, -8.437499046999999 -5.510747428, -8.437223274999999
-5.720836693, -8.436960148000001 -5.863118398, -8.43677576 -6.002404266,
-8.436751781 -6.142806693, -8.434144440000001 -8.956582716, -8.434214752000001
-9.026487204, -8.433763834000001 -9.096728881000001, -8.432243388 -11.21056001,
-8.431431442999999 -11.28053469, -8.431239261 -11.70301736, -9.391320903
-16.19831782, -9.479487978 -16.47694119, -9.562946048000001 -17.25528595,
-9.563050549 -17.32317266, -9.563484803 -17.60777515)
```

[41]:



[42]:
```
# contains
print (polygon.contains(linearring))
print (polygon.touches(linearring))
# polygon.intersect(linearring) #### polygon does not have intersect function,␣
↪only lines can intersect
```

```
False
True
```

[43]:
```
# a vertex
point = Point(-9.563484803, -17.60777515)
print (polygon.contains(point))
print (polygon.touches(point))   ### for point on the edge of polygon, it only␣
↪means touch , not contain
# polygon.intersect(point)
```

```
False
True
```

[44]:
```
linearring.contains(point)
```

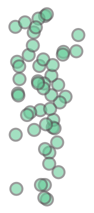
[44]:
```
True
```

4.4 Visualzation

[45]:
```
## generate some random points
# np.random.seed(1)
xy_points_random = np.random.randn(50,2) ; # dimension 250 x 2

xy_points_random[:,0] = xy_points_random[:,0] - 7.5
xy_points_random[:,1] = xy_points_random[:,1] * 3 - 5
```

```
points = MultiPoint(xy_points_random)

points
```

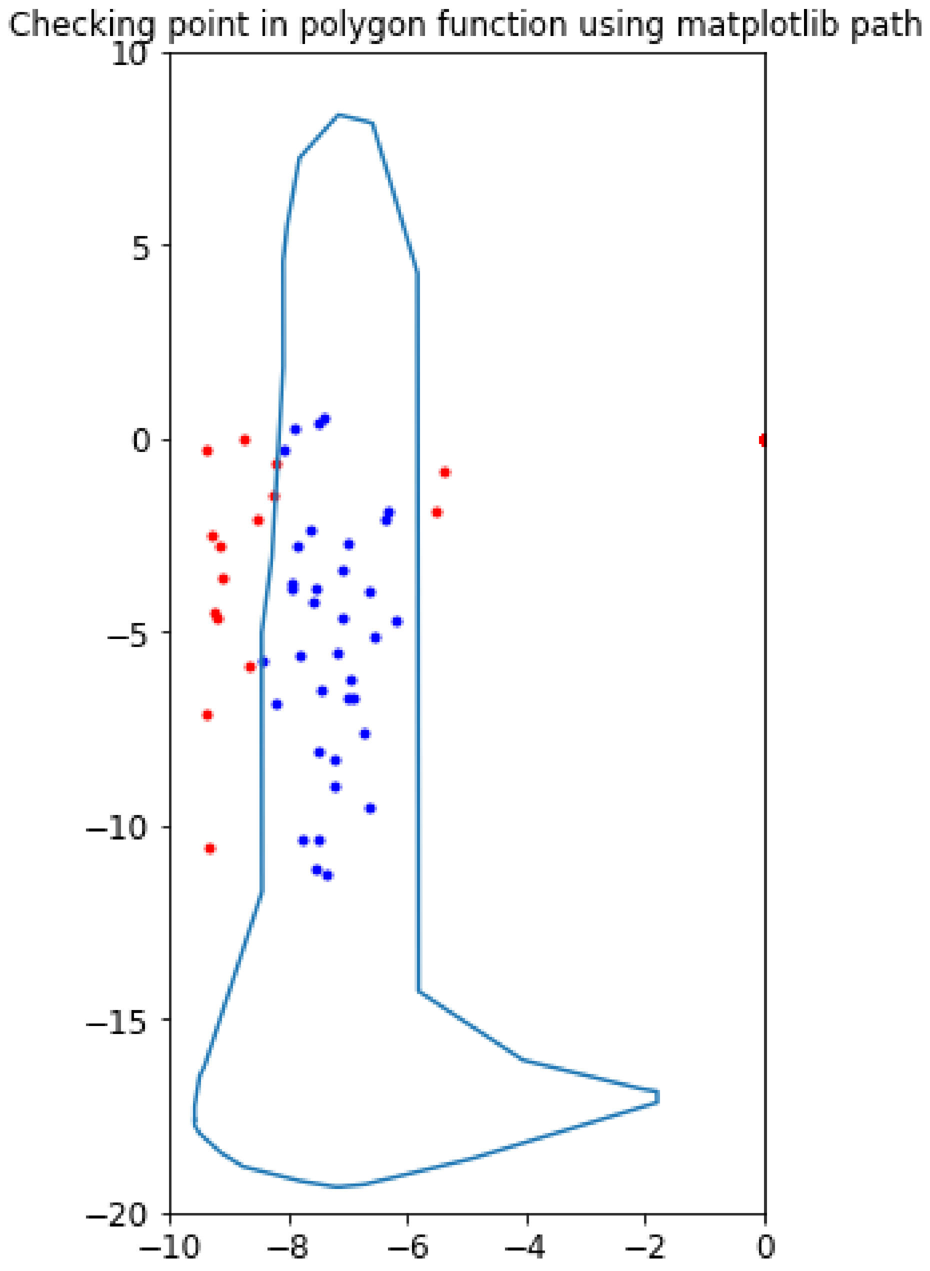
[45]:



[46]:
```
## Determine whether each point lies inside or on the edge of the polygon area.
## --- this will only return one value-----
point_in = polygon.contains(points)
point_on_edge = polygon.touches(points)
##---- the correct way---------
point_in_list = []
for point in points:
    point_in_list.append(polygon.contains(point))

point_in_array = np.array(point_in_list)
print (point_in_list)
```

```
[True, True, True, False, True, True, True, True, True, False, True, True, True,
True, False, True, True, False, False, True, True, True, False, False, False,
True, True, True, True, False, False, True, True, True, True, False, True,
False, True, True, True, False, True, False, True, True, False, True, True,
True]
```

[47]:
```
## visualize the graph
# Plotting the output
plt.figure(figsize=(4, 8))

plt.plot(xy_points_random[:, 0]*point_in_array, xy_points_random[:,␣
 ↪1]*point_in_array, 'b.')
plt.plot(xy_points_random[:, 0]*(~point_in_array), xy_points_random[:,␣
 ↪1]*(~point_in_array), 'r.')

plt.plot(boundary_points[:, 0], boundary_points[:, 1])

plt.xlim([-10, 0])
plt.ylim([-20, 10])
plt.title("Checking point in polygon function using matplotlib path")
plt.show()
```



4.4.1 Summary

- `Due to more robust and user-friendly functionalities, Shapely is chosen`

4.5 Polygon with hole (shapely)

- given **outer** as a plain Polygon and **inners** as a list of plain Polygons (each of them contained in outer) :
- Polygons

class Polygon(shell[, holes=None]) The Polygon constructor takes two positional parameters. The first is an ordered sequence of (x, y[, z]) point tuples and is treated exactly as in the LinearRing case. The second is an optional unordered sequence of ring-like sequences specifying the interior boundaries or ``holes'' of the feature.

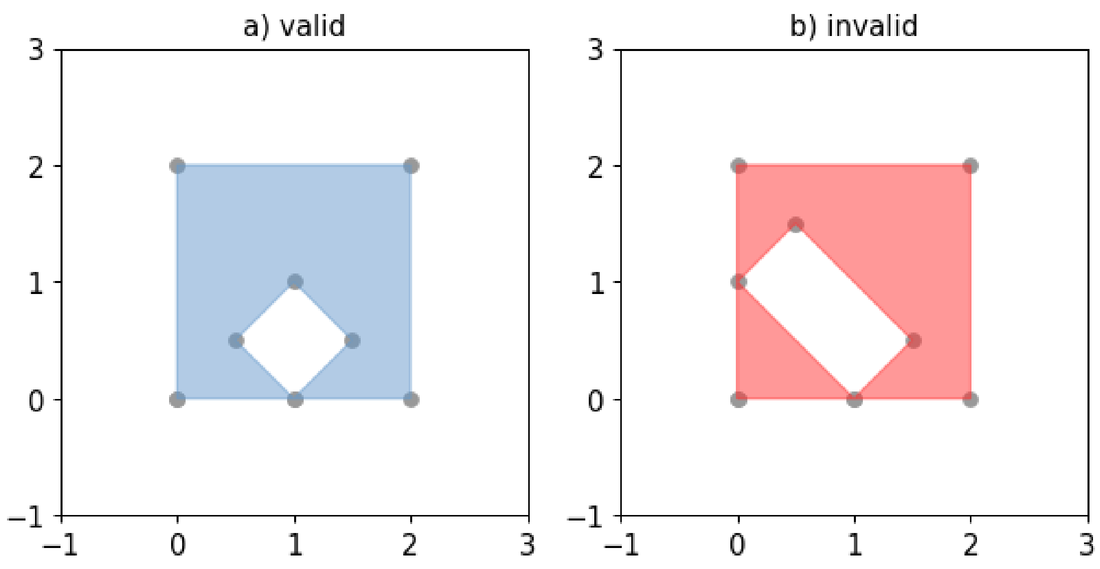
4.6 figures.py – an utility function downloaded from shapely.figures.py

https://raw.githubusercontent.com/Toblerity/Shapely/master/docs/code/figures.py

[48]:
```python
from math import sqrt
from shapely import affinity

GM = (sqrt(5)-1.0)/2.0
W = 8.0
H = W*GM
SIZE = (W, H)

BLUE = '#6699cc'
GRAY = '#999999'
DARKGRAY = '#333333'
YELLOW = '#ffcc33'
GREEN = '#339933'
RED = '#ff3333'
BLACK = '#000000'

COLOR_ISVALID = {
    True: BLUE,
    False: RED,
}

def plot_line(ax, ob, color=GRAY, zorder=1, linewidth=3, alpha=1):
    x, y = ob.xy
    ax.plot(x, y, color=color, linewidth=linewidth, solid_capstyle='round',␣
 ↪zorder=zorder, alpha=alpha)

def plot_coords(ax, ob, color=GRAY, zorder=1, alpha=1):
    x, y = ob.xy
    ax.plot(x, y, 'o', color=color, zorder=zorder, alpha=alpha)

def color_isvalid(ob, valid=BLUE, invalid=RED):
    if ob.is_valid:
        return valid
    else:
        return invalid

def color_issimple(ob, simple=BLUE, complex=YELLOW):
    if ob.is_simple:
        return simple
    else:
        return complex

def plot_line_isvalid(ax, ob, **kwargs):
    kwargs["color"] = color_isvalid(ob)
    plot_line(ax, ob, **kwargs)

def plot_line_issimple(ax, ob, **kwargs):
    kwargs["color"] = color_issimple(ob)
    plot_line(ax, ob, **kwargs)

def plot_bounds(ax, ob, zorder=1, alpha=1):
    x, y = zip(*list((p.x, p.y) for p in ob.boundary))
    ax.plot(x, y, 'o', color=BLACK, zorder=zorder, alpha=alpha)

def add_origin(ax, geom, origin):
    x, y = xy = affinity.interpret_origin(geom, origin, 2)
    ax.plot(x, y, 'o', color=GRAY, zorder=1)
    ax.annotate(str(xy), xy=xy, ha='center',
                textcoords='offset points', xytext=(0, 8))

def set_limits(ax, x0, xN, y0, yN):
    ax.set_xlim(x0, xN)
    ax.set_xticks(range(x0, xN+1))
    ax.set_ylim(y0, yN)
    ax.set_yticks(range(y0, yN+1))
    ax.set_aspect("equal")
```

[49]:
```
from matplotlib import pyplot
from shapely.geometry import Polygon
from descartes.patch import PolygonPatch


fig = pyplot.figure(1, figsize=SIZE, dpi=90)

# 1: valid polygon
ax = fig.add_subplot(121)

ext = [(0, 0), (0, 2), (2, 2), (2, 0), (0, 0)]
```

```
int = [(1, 0), (0.5, 0.5), (1, 1), (1.5, 0.5), (1, 0)][::-1]
polygon = Polygon(ext, [int])

plot_coords(ax, polygon.interiors[0])
plot_coords(ax, polygon.exterior)

patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_title('a) valid')

set_limits(ax, -1, 3, -1, 3)

#2: invalid self-touching ring
ax = fig.add_subplot(122)
ext = [(0, 0), (0, 2), (2, 2), (2, 0), (0, 0)]
int = [(1, 0), (0, 1), (0.5, 1.5), (1.5, 0.5), (1, 0)][::-1] # [::-1] CCW␣
 ↪direction
polygon = Polygon(ext, [int])

plot_coords(ax, polygon.interiors[0])
plot_coords(ax, polygon.exterior)

patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_title('b) invalid')

set_limits(ax, -1, 3, -1, 3)

pyplot.show()
```



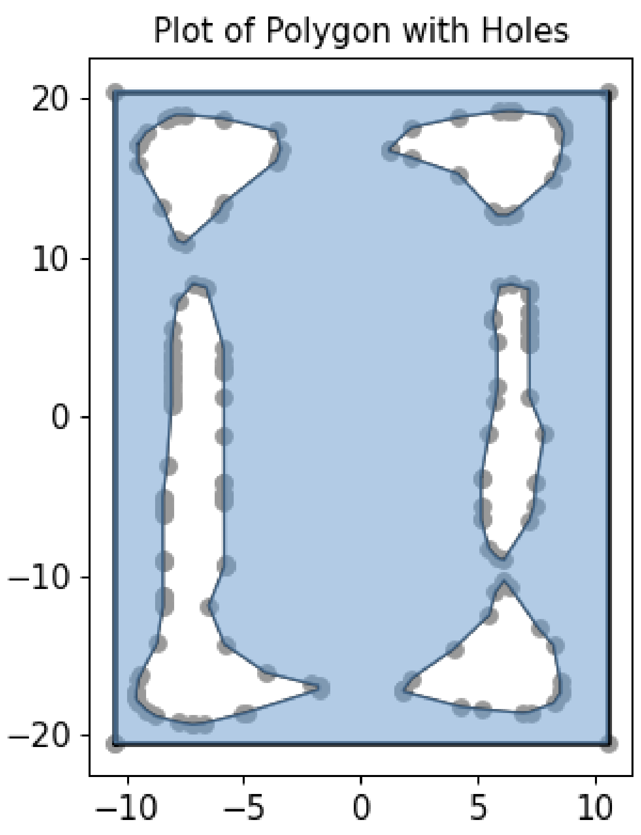
4.7 Testing with our example – Plot the polygon with clustered bulge area as holes

[50]:
```
def plot_filled_polygons(polygons, facecolour='green', edgecolour='black',␣
↪linewidth=1, alpha=0.5):
        """
        This function plots a series of shapely polygons but fills them in

        Args:
            ax_list: list of axes
            polygons: list of shapely polygons

        Author: FJC
        """
        from shapely.geometry import Polygon
        from descartes import PolygonPatch
        from matplotlib.collections import PatchCollection

        print('Plotting the polygons...')

        #patches = []
        for key, poly in polygons.items():
            this_patch = PolygonPatch(poly, fc=facecolour, ec=edgecolour,␣
↪alpha=alpha)
            self.ax_list[0].add_patch(this_patch)
```

[51]:
```
# print (boundary_points0)
```

[52]:
```
fig = pyplot.figure(1, figsize=SIZE, dpi=90)

# 1: valid polygon
ax = fig.add_subplot(121)

# Define exterior boundary CCW direction
ext = [(-10.5, -20.5), (10.5, -20.5), (10.5, 20.5), (-10.5, 20.5), (-10.5, -20.
 ↪5)]
## Define interior boundaries
## ---- boundary_points0, boundary_points1, boundary_points ,boundary_points3,␣
 ↪boundary_points4
# boundary_points0 = boundary_points0[::-1]
polygon = Polygon(ext, [boundary_points0[::-1], boundary_points1[::-1],␣
 ↪boundary_points2[::-1], boundary_points3[::-1], boundary_points4[::-1]])

plot_coords(ax, polygon.interiors[0])
plot_coords(ax, polygon.interiors[1])
plot_coords(ax, polygon.interiors[2])
plot_coords(ax, polygon.interiors[3])
plot_coords(ax, polygon.interiors[4])
plot_coords(ax, polygon.exterior)

plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=2, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)

patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_title('Plot of Polygon with Holes')
```

[52]:
```
Text(0.5, 1.0, 'Plot of Polygon with Holes')
```



[53]:
```
## Testing the points
myPoint = Point(-7.5, -18)
print(myPoint.within(polygon))  # returns 'False'
myPoint2 = Point(0, 0)
print(myPoint2.within(polygon))  # returns 'True'
myPoint  = Point(5, 0)
print(myPoint3.within(polygon))  # returns 'True'
```

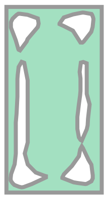
```
False
True
True
```

[54]:
```
list(polygon.exterior.coords)
```

[54]:
```
[(-10.5, -20.5), (10.5, -20.5), (10.5, 20.5), (-10.5, 20.5), (-10.5, -20.5)]
```

[55]:
```
# jupyter notebook is able to visualize directly
polygon
```

[55]:



```
[56]: # list(polygon.interiors[0].coords)
```

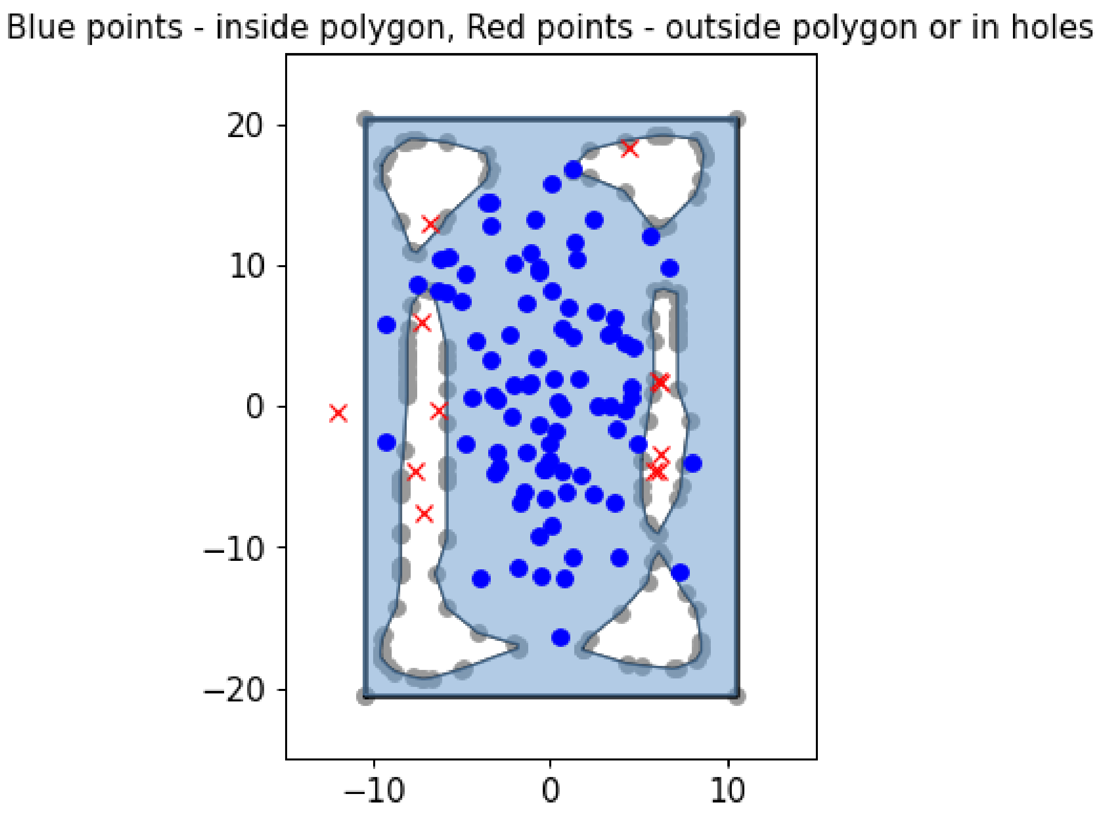
4.7.1 Check Points in holes and polygon

```
[57]: ## generate some random points
# np.random.seed(1)
xy_points_random = np.random.randn(100,2) ; # dimension 250 x 2

xy_points_random[:,0] = xy_points_random[:,0]*4
xy_points_random[:,1] = xy_points_random[:,1]*8

points = MultiPoint(xy_points_random)


## Determine whether each point lies inside or on the edge of the polygon area.
##---- the correct way---------
point_in_list = []
for point in points:
    point_in_list.append(polygon.contains(point))

point_out_list = []
for point in points:
    point_out_list.append(~polygon.contains(point))

# point_touch_list = []
# for point in points:
#     point_touch_list.append(polygon.touches(point))

# for point in points:
#     point_in_list.append(point.within(polygon))

point_in_array = np.array(point_in_list)
point_out_array = np.array(point_out_list)

#----visualization
fig = pyplot.figure(1, figsize=SIZE, dpi=90)
```

```
ax = fig.add_subplot(121)

# Define exterior boundary CCW direction
ext = [(-10.5, -20.5), (10.5, -20.5), (10.5, 20.5), (-10.5, 20.5), (-10.5, -20.
 ↪5)]
## Define interior boundaries
## ---- boundary_points0, boundary_points1, boundary_points ,boundary_points3,␣
 ↪boundary_points4
# polygon = Polygon(ext, [boundary_points0[::-1], boundary_points1[::-1],␣
 ↪boundary_points2[::-1], boundary_points3[::-1], boundary_points4[::-1]])

plot_coords(ax, polygon.interiors[0])
plot_coords(ax, polygon.interiors[1])
plot_coords(ax, polygon.interiors[2])
plot_coords(ax, polygon.interiors[3])
plot_coords(ax, polygon.interiors[4])
plot_coords(ax, polygon.exterior)

plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=2, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)

i = 0
for point in point_in_array:
    if point == True:
        plt.plot(xy_points_random[i, 0], xy_points_random[i, 1], 'bo')
    else:
        plt.plot(xy_points_random[i, 0], xy_points_random[i, 1], 'rx')
    i  = i+1

# plt.plot(xy_points_random[:, 0]*point_in_array, xy_points_random[:,␣
 ↪1]*point_in_array, 'bo')
# plt.plot((xy_points_random[:, 0]*(~point_in_array))  , (xy_points_random[:,␣
 ↪1]*(~point_in_array)) , 'rx')


patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
```

```
ax.add_patch(patch)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Blue points - inside polygon, Red points - outside polygon or in␣
 ↪holes')
```

[57]:
```
Text(0.5, 1.0, 'Blue points - inside polygon, Red points - outside polygon or in
holes')
```



5 2. Line In Polygon

5.0.1 – looks at a straight line and a closed polygon and determines which segments of the line are located inside the polygon.

5.0.2 MatLab Equivalent:

- `[anyIn, inSegment, outSegment] = lineinpolygon(x1, y1, x2, y2, xv, yv)`

- Input variables:
  - x1: x coordinate of line starting point
  - y1: y coordinate of line starting point
  - x2: x coordinate of line ending point
  - y2: y coordinate of line ending point
  - xv: x coordinates of vertices of polygon
  - yv: y coordinates of vertices of polygon
- Output variables:
  - anyIn: true if any portion of the line is located in the polygon
  - inSegment: n x 2 [x y] array of line segments located inside the polygon. Segments are separated by NaNs.
  - outSegment: n x 2 [x y] array of line segments located outside the polygon. Segments are separated by NaNs.

5.1 2.1 determine the line and polygon intersection point

And also the vertices of the edge of the polygon where line intersects

https://gis.stackexchange.com/questions/339409/find-the-vertices-of-the-edge-of-the-polygon-whe

[58]:
```
from shapely.wkt import loads
lin = loads('LineString (289.63171806167395061 -200.22555066079294761, 380.
↪69030837004402201 -65.28898678414094547)')
pol = loads('Polygon ((112.23259911894263041 -229.94933920704846742, 178.
↪75726872246687549 -113.4132158590308137, 309.44757709251092592 -114.
↪35682819383258391, 376.44405286343607031 -230.42114537444933831, 305.
↪67312775330390195 -344.59823788546259493, 176.39823788546246419 -345.
↪07004405286346582, 112.23259911894263041 -229.94933920704846742))')

# the LinearRing
from shapely.geometry import LineString
polin = LineString(list(pol.exterior.coords))

# intersection
pt = polin.intersection(lin)
print(pt.wkt)
```

```
POINT (327.0268317294637 -144.8110298888352)
```

[59]:
```
from shapely.geometry import Point
point = Point(327.0268317294637,-144.8110298888352)
point
```

[59]:

- iterate through the edges of the LinearRing of the polygon
- Now using Determine if Shapely point is within a LineString/MultiLineString

(using the answer of Mike T using the distance with an appropriate threshold because there are floating point precision errors when finding a point on a line)

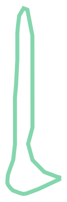
[60]:
```python
# iterate through the edges to determine if Shapely point is within
points = list(polin.coords)
for i,j in zip(points, points[1:]):
   if LineString((i,j)).distance(pt) < 1e-8:
       print(i,j)
# (309.4475770925109, -114.35682819383258) (376.44405286343607, -230.
↪42114537444934)
```

```
(309.4475770925109, -114.35682819383258) (376.44405286343607,
-230.42114537444934)
```

5.1.1 Implementation in our scenario

[61]:
```python
line_example = LineString([(-5, -20), (-5.1, 20)])

poly_lines_exterior = LineString(list(polygon.exterior.coords))
poly_lines_holes_0 = LineString(list(polygon.interiors[0].coords))
poly_lines_holes_1 = LineString(list(polygon.interiors[1].coords))
poly_lines_holes_0
```

[61]:



[62]:
```python
intersection_point = poly_lines_holes_0.intersection(line_example)
print(intersection_point.wkt)
intersection_point
```

```
MULTIPOINT (-5.012282407186927 -15.08703712522939, -5.003388029421077
-18.64478823156891)
```
[62]:

[63]:
```
from shapely.geometry import MultiLineString
# poly_lines = MultiLineString
# poly_lines = [poly_lines_holes_0, poly_lines_holes_1]


# intersection_point = poly_lines.intersection(line_example)  ## this does not␣
↪work
# print(intersection_point.wkt)
# intersection_point
print(list(polygon.exterior.coords))
tuple(list(polygon.exterior.coords))
```

```
[(-10.5, -20.5), (10.5, -20.5), (10.5, 20.5), (-10.5, 20.5), (-10.5, -20.5)]
```

[63]:
```
((-10.5, -20.5), (10.5, -20.5), (10.5, 20.5), (-10.5, 20.5), (-10.5, -20.5))
```

[64]:
```
poly_lines = [1]
poly_lines = list(polygon.exterior.coords)

for interior in polygon.interiors:
    poly_lines.append(list(interior.coords))
#     print (list(interior.coords))
#     print ("\n")
```

[65]:
```
polygon_string = polygon.boundary
polygon_string  ## type is MultiLineString
print (polygon_string)
```

```
MULTILINESTRING ((-10.5 -20.5, 10.5 -20.5, 10.5 20.5, -10.5 20.5, -10.5 -20.5),
(-9.563484803 -17.60777515, -9.563050549 -17.32317266, -9.562946048000001
-17.25528595, -9.479487978 -16.47694119, -9.391320903 -16.19831782,
-8.692026223999999 -14.24323277, -8.430800783 -11.9150651, -8.431330598000001
-11.84282633, -8.431104618999999 -11.77279364, -8.431239261 -11.70301736,
-8.431431442999999 -11.28053469, -8.432243388 -11.21056001, -8.433763834000001
-9.096728881000001, -8.434214752000001 -9.026487204, -8.434144440000001
-8.956582716, -8.436751781 -6.142806693, -8.43677576 -6.002404266,
-8.436960148000001 -5.863118398, -8.437223274999999 -5.720836693,
-8.437499046999999 -5.510747428, -8.437711748 -5.22867673, -8.437665414
-5.018369815, -8.266402065999999 -3.055054607, -8.075370502 0.73404204,
-8.075564515 0.946237627, -8.075723902 1.155397275, -8.076342351999999
1.296058663, -8.076223816000001 1.505751217, -8.076762302000001 1.858143052,
-8.077243371 2.347575652, -8.077373380999999 2.418044374, -8.077292632000001
2.62974456, -8.077797985 2.979395178, -8.078126127000001 3.260022964,
-8.078465016999999 3.61152406, -8.078297421 3.820586477, -8.078231165
3.892557185, -8.079186519 4.524080346, -8.078796486 4.594241844,
-8.010910299000001 5.503780578, -7.805423417 7.253158296, -7.150630623
8.372160513000001, -6.740901329 8.223440951000001, -6.575834436
8.151917065999999, -5.824166189 4.277974596, -5.823328785 3.576119223,
-5.822759678 3.084029174, -5.822809018 2.943934876, -5.822886251 2.873642357,
-5.821247198 1.259605659, -5.819071757 -1.126387678, -5.815524175 -4.076463852,
-5.816269199 -4.146189281, -5.815186309 -4.847855625, -5.815240292 -4.988662938,
-5.814532519 -5.200861489, -5.814904248 -5.340867539, -5.810989539 -9.280515926,
-5.81097803 -9.350380465000001, -5.810725621 -9.422817607000001, -6.467404238
-11.87138807, -5.80649212 -14.28080698, -4.051363424 -16.0624997, -2.086718422
-16.79918684, -1.799246999 -16.87584928, -1.798783509 -17.01800267, -1.798438623
-17.15817805, -4.882875368 -18.5976883, -5.062458582 -18.66787476, -6.715897721
-19.27917588, -7.124066844 -19.34029991, -7.20933058 -19.33870567, -7.779923279
-19.19316765, -8.751138219 -18.81763292, -9.117611514 -18.46083308,
-9.478516986000001 -17.95959273, -9.563020136 -17.74887974, -9.563484803
-17.60777515), (-9.511157609 17.12312109, -9.422013406 17.33394528, -9.151587392
17.88949919, -8.287318673 18.72359183, -8.092376368 18.86245815, -7.863647886
19.00145195, -7.512293324 18.99059348, -5.837142703 18.76033372, -3.59152755
17.95838493, -3.406188026 16.83536133, -3.497607694 16.4095063, -3.589682652
16.05817396, -5.831945135 13.4122299, -6.048540248 12.8521824, -7.504225093
10.9021958, -7.808281653 11.04733916, -7.856092606 11.12160241, -8.454567181
13.16732324, -9.509909097 15.92699988, -9.511157609 17.12312109), (1.204575168
16.77471972, 1.20548297 16.84659083, 2.144147036 18.17436726, 4.200885109
18.84873654, 5.891275394 19.18469623, 6.245575378 19.2499589, 6.511787674
19.24727959, 8.247212158 18.94355361, 8.392701411999999 18.73729733, 8.577848864
18.23441704, 8.66716664 17.88435272, 8.667270354999999 17.74265523, 8.667165767
17.67507386, 8.491482762 16.04937152, 8.189348513000001 14.99510917, 6.517683261
12.83792935, 6.25190999 12.63064761, 5.897635579 12.6362975, 5.810300086
12.64084099, 5.723533601 12.77801468, 5.637645568 12.91955641, 4.147992904
15.26211891, 2.14603108 16.27315717, 1.205106715 16.70453767, 1.204575168
16.77471972), (1.832818725 -17.20413106, 1.832887643 -17.13421663, 2.176211911
-16.43816938, 3.936793788 -14.62413069, 5.469916991 -12.46220909, 5.746618304
-10.98454376, 6.11051199 -10.28934134, 6.340445163 -10.64266684, 7.579148839
-13.26570792, 8.217543945999999 -14.33755904, 8.522295013999999 -16.66937287,
8.522607927999999 -17.09178682, 8.522616632 -17.37297751, 8.22068885
-18.00752601, 7.215039965 -18.55908345, 6.904310133 -18.6177727, 5.204718778
-18.38674281, 4.291318681 -18.23168075, 1.833106324 -17.27859482, 1.832818725
-17.20413106), (5.132801519 -3.794329233, 5.458965387 -1.061881461, 5.734846431
0.972073637, 5.820457946 1.95678652, 5.818102596 4.694725196, 5.64474885
6.02850673, 5.644297932 6.098748407, 5.643807691 6.168940442, 5.643953441
```

```
6.238913801, 5.901958532 8.204429813000001, 6.424415808 8.337073518, 7.189678206
8.045898308, 7.190207832 7.764075072, 7.189951006 7.623454697, 7.190557805
6.64127, 7.191330298 6.569218185, 7.191558518 5.937466182, 7.191414173
5.867503443, 7.191943052 5.588052181, 7.1925382 5.376142971, 7.192384056
4.883528325, 7.192624615 4.67205548, 7.192753626 4.60158672, 7.192724081
4.534084207, 7.196439205 1.301194357, 7.834778857 -1.02114569, 7.480621199
-4.177673763, 7.393734281 -5.652073647, 7.203143769 -6.500492099, 6.109135227
-9.021742599, 5.829844005 -8.86960755, 5.46552895 -8.306438842, 5.135262039
-6.466418378, 5.135173771 -6.396957836, 5.134386919 -5.553277099, 5.132801519
-3.794329233))
```

[66]:
```python
## Bounding box
polygon_string.bounds
```

[66]:
```
(-10.5, -20.5, 10.5, 20.5)
```

[67]:
```python
polygon.bounds
```

[67]:
```
(-10.5, -20.5, 10.5, 20.5)
```

Access the members of a MultiLineString object

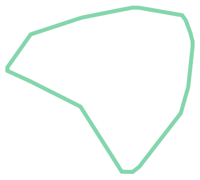
- Its members are instances of LineString and are accessed via the geoms property or via the iterator protocol using in or list().

[68]:
```python
len(polygon_string.geoms)
```

[68]:
```
6
```

[69]:
```python
polygon_string.geoms[3]
```

[69]:



5.1.2 Find intersection point for a random line with the given polygon with holes

[70]:
```python
line_example = LineString([(5, -23), (5.1, 23)])
# line_example2 = LineString([(5, -23), (5, 23)])

points = []
for lines in polygon_string.geoms:
    if lines.intersection(line_example):
        intersection_point = lines.intersection(line_example) ## returns␣
↪MultiPoints object
        print (intersection_point)
        for point in intersection_point.geoms:
            points.append(point)


points = tuple(points)
intersection_multipoint = MultiPoint(points)
print(intersection_multipoint)
intersection_multipoint
```

```
MULTIPOINT (5.005434782608695 -20.5, 5.094565217391304 20.5)
MULTIPOINT (5.079992385580981 13.79649736725153, 5.091360251393237
19.0257156408892)
MULTIPOINT (5.010100644041152 -18.35370374107048, 5.021533710350291
-13.09449323886593)
MULTIPOINT (5.005434782608695 -20.5, 5.094565217391304 20.5, 5.079992385580981
13.79649736725153, 5.091360251393237 19.0257156408892, 5.010100644041152
-18.35370374107048, 5.021533710350291 -13.09449323886593)
```

[70]:

[71]:
```
len(polygon_string.geoms)
```

[71]: 6

5.2 2.2 Line polygon segmentation

- Developing the python function for line in polygon

function [anyIn, inSegment, outSegment] = lineinpolygon(x1, y1, x2, y2, xv, yv)

isLeftToRight = x1 < x2;

if iscell(xv) [xv, yv] = polyjoin(xv, yv); end

[xint, yint] = polyxpoly([x1 x2], [y1 y2], xv, yv, 'unique'); intPoints = [xint yint]; if isempty(intPoints) if isLeftToRight intPoints = sortrows(intPoints, 1); -- sorts the rows of intPoints in ascending order based on the elements

```
in the first column else intPoints = flipud(sortrows(intPoints, 1)); end end
nsegments = size(intPoints,1) + 1;

xin = []; yin = []; xout = []; yout = [];

segmentIsIn = inpolygons(x1, y1, xv, yv);

for iseg = 1:nsegments if iseg == 1 xseg1 = x1; yseg1 = y1; else xseg1 =
intPoints(iseg-1,1); yseg1 = intPoints(iseg-1,2); end if iseg == nsegments xseg2
= x2; yseg2 = y2; else xseg2 = intPoints(iseg,1); yseg2 = intPoints(iseg,2); end
if segmentIsIn xin = [xin xseg1 xseg2 NaN]; yin = [yin yseg1 yseg2 NaN]; else
xout = [xout xseg1 xseg2 NaN]; yout = [yout yseg1 yseg2 NaN]; end segmentIsIn =
segmentIsIn; end

anyIn =  isempty(xin); inSegment = [xin' yin']; outSegment = [xout' yout'];
```

[72]:
```
def LinePolygonIntersectionPoints(input_line, polygon):
    '''
    Functionality:
        Return the line and polygon intersection points

    Input variable:
        - line: shapely LineString object, example LineString([(x1, y1), (x2,␣
↪y2)])
        - polygon: shapely Polygon object, contains exterior and interiors␣
↪vertices (holes)

        example: list(polygon.exterior.coords)
             [(-10.5, -20.5), (10.5, -20.5), (10.5, 20.5), (-10.5, 20.5), (-10.
↪5, -20.5)]
        polygon = Polygon(ext, [boundary_points0[::-1], boundary_points1[::-1],␣
↪boundary_points2[::-1], boundary_points3[::-1], boundary_points4[::-1]])


    Output:
        - intersection_multipoint: a MultiPoint object containing all the␣
↪intersection point.

    '''



    #---------STEP 1:--------------------
    #---Find Intersection Point----------
    #------------------------------------
    ## empty list to store the intersection point,
    ## a list of Point object
    list_points = []

    # -------get the boundary points of the polygon ----------------
    polygon_string = polygon.boundary  ## type is MultiLineString

    # iterate each edge lines in the polygon
    for edges in polygon_string.geoms:
        if edges.intersection(input_line): # if intersection point exists
            intersection_point = edges.intersection(input_line) ##␣
↪intersection_point is a MultiPoints object
#             print (intersection_point)
            # iterate each point in this MultiPoint object
            for point in intersection_point.geoms:
                # append the point to the list
                list_points.append(point)

    # convert the point list to a tuple
    list_points = tuple(list_points)
    # create a MultiPoint Object to return all the intersection points
    intersection_multipoint = MultiPoint(list_points)

    print(intersection_multipoint)

    return intersection_multipoint
```

[73]:
```
intersection_points = LinePolygonIntersectionPoints(line_example, polygon)
intersection_points
```

```
MULTIPOINT (5.005434782608695 -20.5, 5.094565217391304 20.5, 5.079992385580981
13.79649736725153, 5.091360251393237 19.0257156408892, 5.010100644041152
-18.35370374107048, 5.021533710350291 -13.09449323886593)
```

[73]:

[74]:
```
# intersection_points_xxx = LinePolygonIntersectionPoints(line_example3,␣
↪polygon)
```

[75]:
```
# intersection_points_xxx.geoms
```

[76]:
```
intersection_points.geoms[0]
print(intersection_points.geoms[0])
```

```
POINT (5.005434782608695 -20.5)
```

[77]:
```
polygon.interiors[0].coords
```

[77]:
```
<shapely.coords.CoordinateSequence at 0x7f8530504128>
```

[78]:
```
intersection_points.geoms[0].coords
```

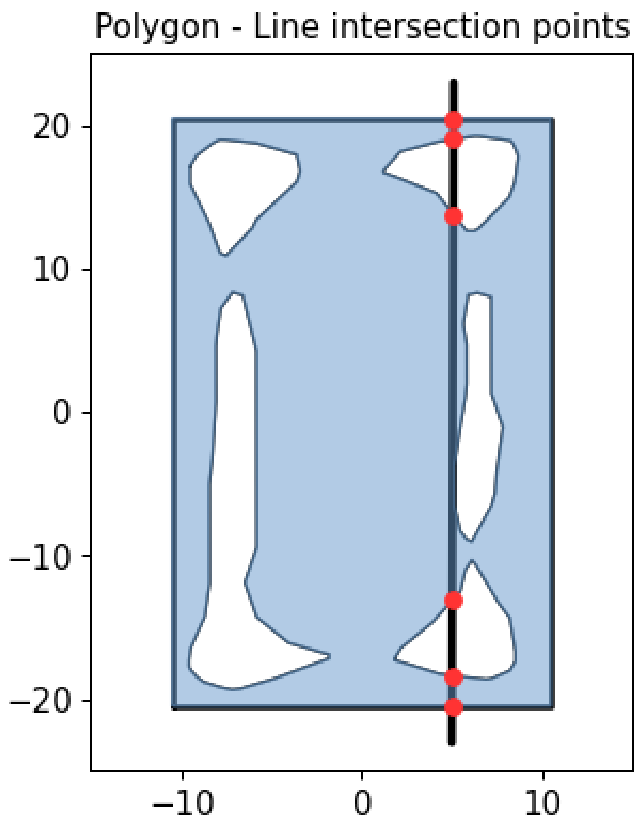
[78]:
```
<shapely.coords.CoordinateSequence at 0x7f8530504160>
```

5.2.1 visualize polygon - line intersection

[79]:
```
#----visualization
fig = pyplot.figure(1, figsize=SIZE, dpi=90)
ax = fig.add_subplot(121)


## Define interior boundaries
## ---- boundary_points0, boundary_points1, boundary_points ,boundary_points3,␣
 ↪boundary_points4
# polygon = Polygon(ext, [boundary_points0[::-1], boundary_points1[::-1],␣
 ↪boundary_points2[::-1], boundary_points3[::-1], boundary_points4[::-1]])
##--------------------------plot the polygon and its␣
 ↪boundary----------------------------------------------------
# plot_coords(ax, polygon.interiors[0])
# plot_coords(ax, polygon.interiors[1])
# plot_coords(ax, polygon.interiors[2])
# plot_coords(ax, polygon.interiors[3])
# plot_coords(ax, polygon.interiors[4])
# plot_coords(ax, polygon.exterior)

plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=2, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)


#----------------------plot the example␣
 ↪line-----------------------------------------------------------
plot_line(ax, ob=line_example, color=BLACK, zorder=1, linewidth=3, alpha=1)
```

```
#----------------------plot the intersection␣
↪point-----------------------------------------------------------
xs = [point.x for point in intersection_points.geoms]
ys = [point.y for point in intersection_points.geoms]
ax.plot(xs, ys, 'o', color=RED, zorder=10, alpha=1)
# ax.plot(intersection_points, 'o', color=RED, zorder=1, alpha=1)


patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Polygon - Line intersection points')
```

[79]:
```
Text(0.5, 1.0, 'Polygon - Line intersection points')
```



[80]:
```
line_example.coords[1]
```

[80]:
```
(5.1, 23.0)
```

[81]:
```
line_example.coords
```

[81]:
```
<shapely.coords.CoordinateSequence at 0x7f8530509128>
```

[82]:
```
intersection_points.geoms[0].coords[:]
list(intersection_points.geoms[0].coords) # equivalent
```

[82]:
```
[(5.005434782608695, -20.5)]
```

[83]:
```
line_seg1 = LineString([intersection_points.geoms[0],intersection_points.
 ↪geoms[1]])
line_seg1
```

[83]:

5.3 sorts the rows of intPoints in ascending order based on the elements in the first column

https://www.kite.com/python/answers/how-to-sort-the-rows-of-a-numpy-array-by-a-column-in-pytho

[84]:
```
list_point_coord = []

for point in intersection_points.geoms:
    list_point_coord.append(point.coords[0])

array_point_coord = np.array(list_point_coord)
print (array_point_coord)
print (array_point_coord.shape)
```

```
[[  5.00543478 -20.5       ]
 [  5.09456522  20.5       ]
 [  5.07999239  13.79649737]
 [  5.09136025  19.02571564]
 [  5.01010064 -18.35370374]
 [  5.02153371 -13.09449324]]
(6, 2)
```

```
[85]: # sort the array based on the second colum -- ascending order
      sorted_point = array_point_coord[np.argsort(array_point_coord[:, 1])]
      # sorted_point = array_point_coord.argsort(axis = 0)
      ## for descending order:
      ## array.argsort()[::-1]

      print (sorted_point)
      print (sorted_point.shape)
```

```
[[  5.00543478 -20.5       ]
 [  5.01010064 -18.35370374]
 [  5.02153371 -13.09449324]
 [  5.07999239  13.79649737]
 [  5.09136025  19.02571564]
 [  5.09456522  20.5       ]]
(6, 2)
```

```
[86]: ## Convert the sorted array into the MultiPoint
      list_point_coord = []

      for point in sorted_point:
      #     print (point) ## each point is a list
      #     print (tuple(point))
          point_ = Point (tuple(point))
          list_point_coord.append(point_)

      print (list_point_coord)

      sorted_multi_point = MultiPoint(tuple(list_point_coord))
      sorted_multi_point
```

```
[<shapely.geometry.point.Point object at 0x7f8530576748>,
<shapely.geometry.point.Point object at 0x7f85305762e8>,
<shapely.geometry.point.Point object at 0x7f853327d390>,
<shapely.geometry.point.Point object at 0x7f853327de80>,
<shapely.geometry.point.Point object at 0x7f8530801be0>,
<shapely.geometry.point.Point object at 0x7f8530801c18>]
```

[86]:

5.3.1 Implement the sorting multi point function

[87]:
```
def sort_MultiPoint(multipoint, ascending = True, axis = 1):
    '''
    This function sort the Shapely MultiPoint Object

    Input/argument:
        - multipoint: a shapely multipoint object
        - ascending: Ture or False (descending order)
        - axis 0 or 1 or None, optional, default 1
          (sort the array based on the first or second colum)


    Return : the multipoint object sorted
    '''

    # an temp empty list storing each points' (x,y) coordinate
    list_point_coord = []

    for point in multipoint.geoms:
        list_point_coord.append(point.coords[0]) # append the coordinate to the␣
 ↪list

    # convert the list to numpy array
    array_point_coord = np.array(list_point_coord)
#     print (array_point_coord)
#     print (array_point_coord.shape)

    if (ascending==True):
        if (axis == 1):
            # sort the array based on the second colum -- ascending order
            sorted_point = array_point_coord[np.argsort(array_point_coord[:,␣
 ↪1])]
        if (axis == 0):
            sorted_point = array_point_coord[np.argsort(array_point_coord[:,␣
 ↪0])]
    else:
        if (axis == 1):
            # sort the array based on the second colum -- descending order
            sorted_point = array_point_coord[np.argsort(array_point_coord[:,␣
 ↪1])[::-1]]
        if (axis == 0):
            sorted_point = array_point_coord[np.argsort(array_point_coord[:,␣
 ↪0])[::-1]]


    ## Convert the sorted array into the MultiPoint
    list_point_coord = []

    for point in sorted_point:
    #     print (point) ## each point is a list
        point_ = Point (tuple(point)) # convert each point to a Shapely Point␣
↪object
        list_point_coord.append(point_)

#     print (list_point_coord)

    sorted_multi_point = MultiPoint(tuple(list_point_coord))
    return sorted_multi_point
```

[88]:
```
sorted_multi_point = sort_MultiPoint(intersection_points)
sorted_multi_point
for point in sorted_multi_point.geoms:
    print (point.coords[0])
```

```
(5.005434782608695, -20.5)
(5.0101006440411515, -18.35370374107048)
(5.021533710350291, -13.094493238865931)
(5.079992385580981, 13.796497367251531)
(5.091360251393237, 19.0257156408892)
(5.094565217391304, 20.5)
```

[89]:
```
sorted_multi_point = sort_MultiPoint(intersection_points, ascending = False )
for point in sorted_multi_point.geoms:
    print (point.coords[0])
```

```
(5.094565217391304, 20.5)
(5.091360251393237, 19.0257156408892)
(5.079992385580981, 13.796497367251531)
(5.021533710350291, -13.094493238865931)
(5.0101006440411515, -18.35370374107048)
(5.005434782608695, -20.5)
```

[90]:
```
points_verticle_sort = MultiPoint([(0,0), (1,0.1), (2,-0.2), (3,0.3)])
points_verticle_sort
```

[90]:

```
[91]: sorted_multi_point = sort_MultiPoint(points_verticle_sort, ascending = False,␣
       ↪axis =0 )
      for point in sorted_multi_point.geoms:
          print (point.coords[0])
```

```
(3.0, 0.3)
(2.0, -0.2)
(1.0, 0.1)
(0.0, 0.0)
```

5.3.2 Line in polygon segmentation function

sorting multi point https://gis.stackexchange.com/questions/338460/reversing-coordinates-of-mul

```
[92]: def LineInPolygonSegmentation(input_line, input_polygon):
          '''
          Functionality:
              - to ouput the segment of a straight line inside and outside the polygon


          Input: LineString object representing a straight lines
          Ouput: Segments of lines. InSeg/OutSeg

          '''



          line_segment_in = [] # empty list to store the segment inside polygon
          line_segment_out = [] # empty list to store the segment outside the polygon


          #---Check if line is moving left-----
          #---to right or vice versa-----------
          #-----------------------------------
          if (input_line.coords[0][0] < input_line.coords[1][0]): ### x1 < x2;
              isLeftToRight = True
              isVertical = False
              isRightToLeft = False
          elif ((input_line.coords[0][0] == input_line.coords[1][0])):   ### x1 = x2;
              isLeftToRight = False
              isVertical = True
              isRightToLeft = False
          else:                                                        ## x1 < x2
              isLeftToRight = False
              isVertical = False
              isRightToLeft = True
```

```
    #---------STEP 1:--------------------
    #---Get intersection point-----------
    #-----------------------------------
    # a MultiPoint Object
    intersection_points = LinePolygonIntersectionPoints(input_line,␣
↪input_polygon)

    ### check number of intersection points
    num_points = len(intersection_points.geoms)

    ## sort the multipoints in certain orders
    if (num_points != 0):
        if isLeftToRight:
            # sort the points in x axis, ascending order
            intersection_point_sorted = sort_MultiPoint(intersection_points,␣
↪ascending = True, axis = 0)
        elif isVertical:
            # sort the points in y axis, ascending order
            intersection_point_sorted = sort_MultiPoint(intersection_points,␣
↪ascending = True, axis = 1)
        elif isRightToLeft:
            # sort the points in x axis, descending order
            intersection_point_sorted = sort_MultiPoint(intersection_points,␣
↪ascending = False, axis = 0)

    else:
        return None, None


    number_of_segments = num_points + 1

    #-----------------------------------------------
    # Divide line into in-segemnts and out-segments
    # ----------------------------------------------
    #-----------------------------------------------

    # from the first coordinate
    currentSegmentIsIn = polygon.contains(Point([input_line.coords[0]]))

    for seg_point in range(num_points):
        #  LineString([(5, -23), (5, 23)]) object

        if (seg_point == 0):
            ## if current intersection point is the first point (closest to␣
↪x1,y1)
            current_seg_line = LineString([Point([input_line.coords[0]]),
                                              intersection_point_sorted.
↪geoms[seg_point]])

        else:
            current_seg_line = LineString([intersection_point_sorted.
↪geoms[seg_point-1],
                                              intersection_point_sorted.
↪geoms[seg_point]])

            if currentSegmentIsIn:
                # append current segment into the line_segment_in list
                line_segment_in.append(current_seg_line)
            else:
                line_segment_out.append(current_seg_line)

        currentSegmentIsIn = ~currentSegmentIsIn


    #--------------------------------------------------------------------------
#     ## add the last segment: (from last intersection point to x2, y2)
#     current_seg_line = LineString([intersection_point_sorted.
↪geoms[seg_point],
#                                    Point([input_line.coords[1]])])

#     if currentSegmentIsIn:
#         # append current segment into the line_segment_in list
#         line_segment_in.append(current_seg_line)
#     else:
#         line_segment_out.append(current_seg_line)
    #--------------------------------------------------------------------------


    line_segment_in = tuple(line_segment_in)
    line_segment_out = tuple(line_segment_out)

    line_segment_in_mutiline = MultiLineString(line_segment_in)
    line_segment_out_mutiline = MultiLineString(line_segment_out)
    print (line_segment_in_mutiline)
    print ("\n")
    print (line_segment_out_mutiline)
    print ("\n")

    return line_segment_in_mutiline, line_segment_out_mutiline
```

[93]:
```
line_segment_in_mutiline, line_segment_out_mutiline =␣
↪LineInPolygonSegmentation(line_example, polygon)
```

```
MULTIPOINT (5.005434782608695 -20.5, 5.094565217391304 20.5, 5.079992385580981
13.79649736725153, 5.091360251393237 19.0257156408892, 5.010100644041152
-18.35370374107048, 5.021533710350291 -13.09449323886593)
MULTILINESTRING ((5.005434782608695 -20.5, 5.010100644041152
-18.35370374107048), (5.021533710350291 -13.09449323886593, 5.079992385580981
13.79649736725153), (5.091360251393237 19.0257156408892, 5.094565217391304
20.5))
```

```
MULTILINESTRING ((5.010100644041152 -18.35370374107048, 5.021533710350291
-13.09449323886593), (5.079992385580981 13.79649736725153, 5.091360251393237
19.0257156408892))
```

[94]:
```
line_segment_in_mutiline
```

[94]:

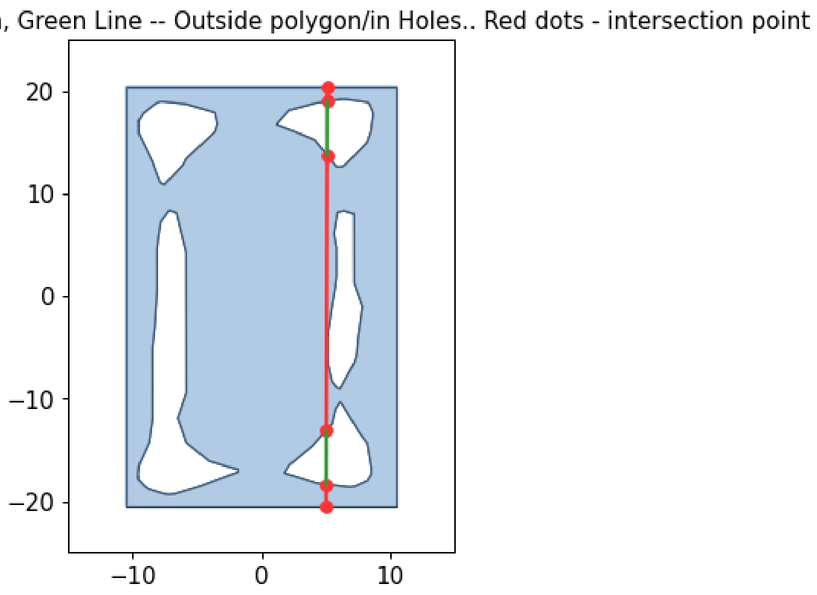
[95]:
```
line_segment_out_mutiline
```

[95]:

[96]:
```
### visualization of the line polygon segmentation

fig = pyplot.figure(1, figsize=SIZE, dpi=90)
ax = fig.add_subplot(121)


plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=1, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
```

```
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)


#----------------------plot the example␣
 ↪line------------------------------------------------------------
# plot_line(ax, ob=line_example, color=BLACK, zorder=1, linewidth=2, alpha=1)

#----------------------plot the intersection␣
 ↪point----------------------------------------------------------
xs = [point.x for point in intersection_points.geoms]
ys = [point.y for point in intersection_points.geoms]
ax.plot(xs, ys, 'o', color=RED, zorder=10, alpha=1)
# ax.plot(intersection_points, 'o', color=RED, zorder=1, alpha=1)


## --------------------plot intersection␣
 ↪lines--------------------------------------------------------------
for i in range (len(line_segment_in_mutiline.geoms)):
    plot_line(ax, ob=line_segment_in_mutiline.geoms[i], color=RED, zorder=10,␣
 ↪linewidth=2, alpha=1)

for i in range (len(line_segment_out_mutiline.geoms)):
    plot_line(ax, ob=line_segment_out_mutiline.geoms[i], color=GREEN,␣
 ↪zorder=10, linewidth=2, alpha=1)


# plot_line(ax, ob=line_segment_in_mutiline.geoms[0], color=GREEN, zorder=10,␣
 ↪linewidth=2, alpha=1)
# plot_line(ax, ob=line_segment_out_mutiline.geoms[0], color=RED, zorder=10,␣
 ↪linewidth=2, alpha=1)
# plot_line(ax, ob=line_segment_out_mutiline.geoms[1], color=RED, zorder=10,␣
 ↪linewidth=2, alpha=1)
# plot_line(ax, ob=line_segment_out_mutiline.geoms[2], color=RED, zorder=10,␣
 ↪linewidth=2, alpha=1)


patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Red Line - Inside polygon, Green Line -- Outside polygon/in Holes.
 ↪. Red dots - intersection point')
```

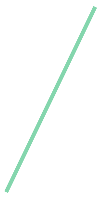
[96]:
```
Text(0.5, 1.0, 'Red Line - Inside polygon, Green Line -- Outside polygon/in
Holes.. Red dots - intersection point')
```



5.4 plot another example line intersection

[97]:
```
line_example2 = LineString ([(-10.1,-21),(10.1,21)])
line_example2
```

[97]:



[98]:
```
intersection_points2  = LinePolygonIntersectionPoints(line_example2, polygon)
line_segment_in_2, line_segment_out_2 =␣
 ↪LineInPolygonSegmentation(line_example2, polygon)
line_segment_out_2
```

```
MULTIPOINT (-9.859523809523807 -20.5, 9.859523809523807 20.5, -8.954949344499807
-18.61920160737584, -6.192165132805279 -12.87479879098127, 6.076088612462458
12.6334515704665, 8.644412896090458 17.97353176414848)
MULTIPOINT (-9.859523809523807 -20.5, 9.859523809523807 20.5, -8.954949344499807
-18.61920160737584, -6.192165132805279 -12.87479879098127, 6.076088612462458
12.6334515704665, 8.644412896090458 17.97353176414848)
MULTILINESTRING ((-9.859523809523807 -20.5, -8.954949344499807
-18.61920160737584), (-6.192165132805279 -12.87479879098127, 6.076088612462458
12.6334515704665), (8.644412896090458 17.97353176414848, 9.859523809523807
20.5))


MULTILINESTRING ((-8.954949344499807 -18.61920160737584, -6.192165132805279
-12.87479879098127), (6.076088612462458 12.6334515704665, 8.644412896090458
17.97353176414848))
```

[98]:

[99]:
```
line_example3 = LineString ([(-11,-21),(-11,21)])
# intersection_points3  = LinePolygonIntersectionPoints(line_example3, polygon)
line_segment_in_3, line_segment_out_3 =␣
 ↪LineInPolygonSegmentation(line_example3, polygon)
line_segment_out_3
```

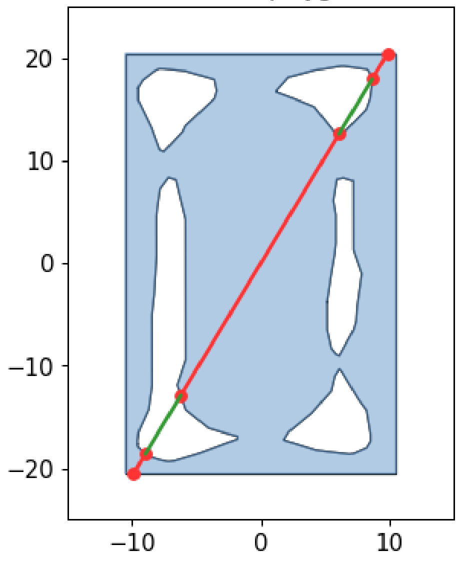
```
GEOMETRYCOLLECTION EMPTY
```

[100]:
```
### visualization of the line polygon segmentation

fig = pyplot.figure(1, figsize=SIZE, dpi=90)
ax = fig.add_subplot(121)



plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=1, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
```

```
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)


#----------------------plot the intersection␣
 ↪point----------------------------------------------------------
xs = [point.x for point in intersection_points2.geoms]
ys = [point.y for point in intersection_points2.geoms]
ax.plot(xs, ys, 'o', color=RED, zorder=10, alpha=1)
# ax.plot(intersection_points, 'o', color=RED, zorder=1, alpha=1)


## --------------------plot intersection␣
 ↪lines-------------------------------------------------------------
for i in range (len(line_segment_in_2.geoms)):
    plot_line(ax, ob=line_segment_in_2.geoms[i], color=RED, zorder=10,␣
 ↪linewidth=2, alpha=1)

for i in range (len(line_segment_out_2.geoms)):
    plot_line(ax, ob=line_segment_out_2.geoms[i], color=GREEN, zorder=10,␣
 ↪linewidth=2, alpha=1)



patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Red Line - Inside polygon, Green Line -- Outside polygon/in Holes.
 ↪. Red dots - intersection point')
```

```
[100]: Text(0.5, 1.0, 'Red Line - Inside polygon, Green Line -- Outside polygon/in
       Holes.. Red dots - intersection point')
```

Red Line - Inside polygon, Green Line -- Outside polygon/in Holes.. Red dots - intersection point



5.5 BufferM2 Algorithm: Computes buffer zone around a polygon

```
polygon.buffer(value)
```

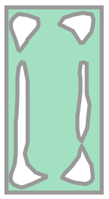
```
value + - make the hole smaller, - make the exterior countour bigger
```

```
value - - make the hole bigger, - make the exterior countour smaller
```

Shapely buffer function

```
[101]: polygon.buffer(0)
```

[101]:



```
[102]: polygon.interiors[0].coords
```

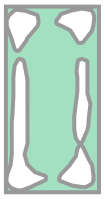
```
[102]: <shapely.coords.CoordinateSequence at 0x7f85304005f8>
```

```
[103]: polygon.exterior.coords[0]
```

```
[103]: (-10.5, -20.5)
```

```
[104]: ## buffer 0.2 unit (not in meters)
       polygon_buffered = polygon.buffer(-0.4)
       polygon_buffered
```

[104]:



```
[105]: polygon_buffered.interiors[0].coords[0]
```

```
[105]: (0.8045866400902585, 16.771690274407504)
```

```
[106]: polygon_buffered.exterior.coords[0]
```

```
[106]: (-10.1, -20.1)
```

5.6 Create Zig-Zag Tool Path

```
[107]: ## Create lines and requried parameters
       delta_x = 1     ## delta_x determines the intervals of lines in x direction

       delta_y = 1000  ## the dx/dy determins the slope of the lines, larger delta_y␣
        ↪gives more vertical lines



       nl = 150 ## % Could probably calculate this
```

```
[108]: # poly_lines = list(polygon.exterior.coords)
       x_countour = tuple([point.x for point in MultiPoint(polygon.exterior.coords)])
       y_countour = tuple([point.y for point in MultiPoint(polygon.exterior.coords)])
```

```
[109]: print (x_countour)
```

```
(-10.5, 10.5, 10.5, -10.5, -10.5)
```

```
[110]: import math

       # MATLAB:
       # xe = floor(min(xc)./dx)*dx + (0:(nl-1))*dx;
       # ye = sort(ceil(max(yc)./dy)*dy - (0:(nl-1))*dy);

```

```
x_extended = []
y_extended = []

# this means 0,1,2,.... nl-1, there are nl numbers in total
for n in range(nl):
    # floor returns floor of x - the largest integer not greater than x
    xe = math.floor(min(x_countour) / delta_x) * delta_x + n * delta_x
    ye = math.ceil(max(y_countour) / delta_y) * delta_y - n *delta_y
    x_extended.append(xe)
    y_extended.append(ye)

y_extended.sort()
```

[111]:
```
print (x_extended)
```

```
[-11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138]
```

[112]:
```
print (y_extended)
```

```
[-148000, -147000, -146000, -145000, -144000, -143000, -142000, -141000,
-140000, -139000, -138000, -137000, -136000, -135000, -134000, -133000, -132000,
-131000, -130000, -129000, -128000, -127000, -126000, -125000, -124000, -123000,
-122000, -121000, -120000, -119000, -118000, -117000, -116000, -115000, -114000,
-113000, -112000, -111000, -110000, -109000, -108000, -107000, -106000, -105000,
-104000, -103000, -102000, -101000, -100000, -99000, -98000, -97000, -96000,
-95000, -94000, -93000, -92000, -91000, -90000, -89000, -88000, -87000, -86000,
-85000, -84000, -83000, -82000, -81000, -80000, -79000, -78000, -77000, -76000,
-75000, -74000, -73000, -72000, -71000, -70000, -69000, -68000, -67000, -66000,
-65000, -64000, -63000, -62000, -61000, -60000, -59000, -58000, -57000, -56000,
-55000, -54000, -53000, -52000, -51000, -50000, -49000, -48000, -47000, -46000,
-45000, -44000, -43000, -42000, -41000, -40000, -39000, -38000, -37000, -36000,
-35000, -34000, -33000, -32000, -31000, -30000, -29000, -28000, -27000, -26000,
-25000, -24000, -23000, -22000, -21000, -20000, -19000, -18000, -17000, -16000,
-15000, -14000, -13000, -12000, -11000, -10000, -9000, -8000, -7000, -6000,
-5000, -4000, -3000, -2000, -1000, 0, 1000]
```

[113]:
```
y_extended[-1] # the last number from this list
```

[113]:
```
1000
```

[114]:
```
# convert the list to array
test_array = np.array([x_extended])
test_array.shape
```

[114]:
```
(1, 150)
```

[115]:
```
# convert to numpy array and also reverse the direction
test_array = np.array([y_extended[::-1]])
test_array.shape
# test_array
```

[115]:
```
(1, 150)
```

[116]:
```
# MATLAB
# xa = xe;
# ya = ones(1,nl).*ye(end);
# xb = ones(1,nl).*xe(1);
# yb = ye(end:-1:1); % Reverse the order of elements

# xl = [xa; xb];
# yl = [ya; yb];

'''
xa, ya gives a horizontal line y=1000, while x ranges from -11 to 137 (interval␣
↪1)

xb, yb gives a vertical line x = -11, while y ranges from 1000 to -147000␣
↪(interval 1000)
'''

xa = np.array([x_extended])
ya = np.ones((1, nl)) * y_extended[-1]
xb = np.ones((1, nl)) * x_extended[0]
yb = np.array([y_extended[::-1]])


xl = np.concatenate((xa, xb))
yl = np.concatenate((ya, yb))
```

[117]:
```
xa
```

[117]:
```
array([[-11, -10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,
          2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,
         15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27,
         28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40,
         41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,
         54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,  66,
         67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,
         80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,
         93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104, 105,
        106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
        119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131,
        132, 133, 134, 135, 136, 137, 138]])
```

```
[118]: xa.shape
```

```
[118]: (1, 150)
```

```
[119]: ya
```

```
[119]: array([[1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000., 1000.,
        1000., 1000., 1000., 1000., 1000., 1000.]])
```

```
[120]: ya.shape
```

```
[120]: (1, 150)
```

```
[121]: xb
```

```
[121]: array([[-11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11., -11., -11., -11., -11.,
        -11., -11., -11., -11., -11., -11., -11.]])
```

[122]:
```
xb.shape
```

[122]:
```
(1, 150)
```

[123]:
```
yb
```

[123]:
```
array([[   1000,       0,   -1000,   -2000,   -3000,   -4000,   -5000,
          -6000,   -7000,   -8000,   -9000,  -10000,  -11000,  -12000,
         -13000,  -14000,  -15000,  -16000,  -17000,  -18000,  -19000,
         -20000,  -21000,  -22000,  -23000,  -24000,  -25000,  -26000,
         -27000,  -28000,  -29000,  -30000,  -31000,  -32000,  -33000,
         -34000,  -35000,  -36000,  -37000,  -38000,  -39000,  -40000,
         -41000,  -42000,  -43000,  -44000,  -45000,  -46000,  -47000,
         -48000,  -49000,  -50000,  -51000,  -52000,  -53000,  -54000,
         -55000,  -56000,  -57000,  -58000,  -59000,  -60000,  -61000,
         -62000,  -63000,  -64000,  -65000,  -66000,  -67000,  -68000,
         -69000,  -70000,  -71000,  -72000,  -73000,  -74000,  -75000,
         -76000,  -77000,  -78000,  -79000,  -80000,  -81000,  -82000,
         -83000,  -84000,  -85000,  -86000,  -87000,  -88000,  -89000,
         -90000,  -91000,  -92000,  -93000,  -94000,  -95000,  -96000,
         -97000,  -98000,  -99000, -100000, -101000, -102000, -103000,
        -104000, -105000, -106000, -107000, -108000, -109000, -110000,
        -111000, -112000, -113000, -114000, -115000, -116000, -117000,
        -118000, -119000, -120000, -121000, -122000, -123000, -124000,
        -125000, -126000, -127000, -128000, -129000, -130000, -131000,
        -132000, -133000, -134000, -135000, -136000, -137000, -138000,
        -139000, -140000, -141000, -142000, -143000, -144000, -145000,
        -146000, -147000, -148000]])
```

[124]:
```
yb.shape
```

[124]:
```
(1, 150)
```

[125]:
```
xl.shape
```

[125]:
```
(2, 150)
```

[126]:
```
yl.shape
```

[126]:
```
(2, 150)
```

5.7 Integrate the grid line creation function

[127]:
```python
def grid_line_creation(polygon, delta_x = 1, delta_y = 1000, nl = 150):
    '''
    Input variables:
        - delta_x determines the intervals of lines in x direction
        - the dx/dy determins the slope of the lines, larger delta_y gives more␣
↪vertical lines
        - nl: a hyperparameter determin the grid points
        - polygon: the input polygon

    Output:
        - line_elements: a list of lines (LineString objects)
    '''

    x_countour = tuple([point.x for point in MultiPoint(polygon.exterior.
↪coords)]) # tuples, x coordinates of the polygon exterior contour
    y_countour = tuple([point.y for point in MultiPoint(polygon.exterior.
↪coords)])

    # two empty list for temperarily store variables
    x_extended = []
    y_extended = []

    # this means 0,1,2,.... nl-1, there are nl numbers in total
    for n in range(nl):
        # floor returns floor of x - the largest integer not greater than x
        xe = math.floor(min(x_countour) / delta_x) * delta_x + n * delta_x
        ye = math.ceil(max(y_countour) / delta_y) * delta_y - n *delta_y
        x_extended.append(xe)
        y_extended.append(ye)
    y_extended.sort() # sort the list in ascending order


    xa = np.array([x_extended])                    # shape is (1, nl)
    ya = np.ones((1, nl)) * y_extended[-1]         # shape is (1, nl)
    xb = np.ones((1, nl)) * x_extended[0]          # shape is (1, nl)
    yb = np.array([y_extended[::-1]])              # shape is (1, nl)


    xl = np.concatenate((xa, xb)) # shape should be (2, nl)
    yl = np.concatenate((ya, yb)) # shape should be (2, nl)


    ## create the LineString element based on the xl yl
    line_elements = []
    for i in range (nl):
        # example: line_element = LineString([(5, -23), (5.1, 23)])
        # xl(1,ii), yl(1,ii), xl(2,ii), yl(2,ii)
        line = LineString([(xl[0,i], yl[0,i]), (xl[1,i], yl[1,i])])
        line_elements.append(line)

    return line_elements
```

[128]:
```
line_elements = grid_line_creation(polygon, delta_x = 1, delta_y = 1000, nl =␣
 ↪150)
line_elements2 = grid_line_creation(polygon, delta_x = 2, delta_y = 1000, nl =␣
 ↪150)
line_elements3 = grid_line_creation(polygon, delta_x = 1, delta_y = 5, nl = 150)
line_elements4 = grid_line_creation(polygon, delta_x = 2, delta_y = 2, nl = 150)
line_elements5 = grid_line_creation(polygon, delta_x = 1, delta_y = 1000, nl =␣
 ↪200)
line_elements6 = grid_line_creation(polygon, delta_x = 2, delta_y = 0.5, nl =␣
 ↪150)
```

[129]:
```
line_elements[1]
```

[129]:

[130]:
```
len(line_elements)
```

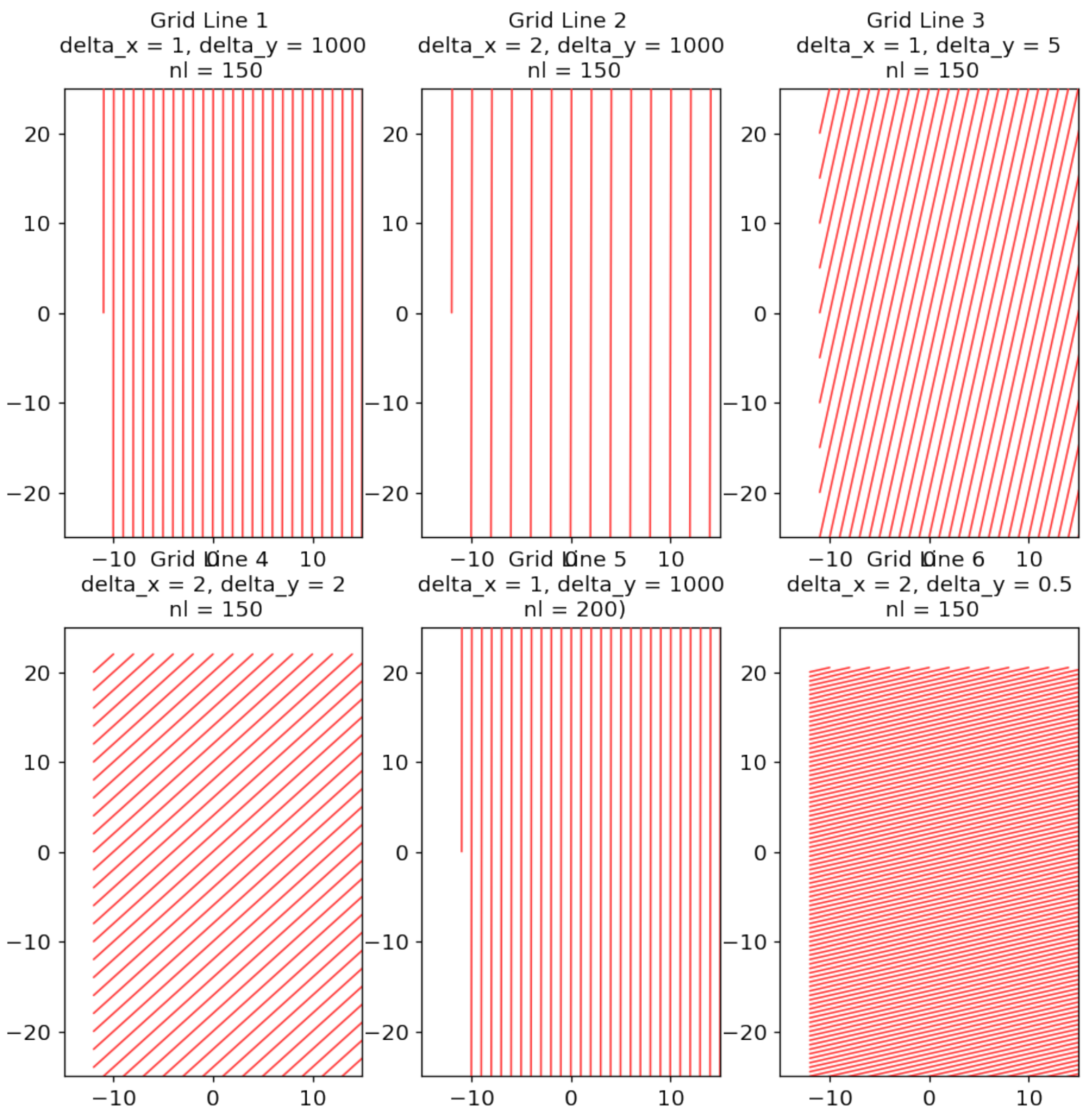
[130]:
```
150
```

5.8 visualize the line grid we created

[131]:
```
### visualization of the line polygon segmentation

fig = pyplot.figure(1, figsize=(10, 10), dpi=110)


ax = fig.add_subplot(231)
## --------------------plot intersection␣
↪lines-------------------------------------------------------------
for i in range (len(line_elements)):
    plot_line(ax, ob=line_elements[i], color=RED, zorder=10, linewidth=1,␣
↪alpha=1)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Grid Line 1 \n delta_x = 1, delta_y = 1000 \n nl = 150')



ax = fig.add_subplot(232)
## --------------------plot intersection␣
↪lines-------------------------------------------------------------
for i in range (len(line_elements)):
    plot_line(ax, ob=line_elements2[i], color=RED, zorder=10, linewidth=1,␣
↪alpha=1)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Grid Line 2 \n delta_x = 2, delta_y = 1000 \n nl = 150')

ax = fig.add_subplot(233)
## --------------------plot intersection␣
↪lines-------------------------------------------------------------
for i in range (len(line_elements)):
    plot_line(ax, ob=line_elements3[i], color=RED, zorder=10, linewidth=1,␣
↪alpha=1)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)
ax.set_title('Grid Line 3 \n delta_x = 1, delta_y = 5 \n nl = 150')
```

```
ax = fig.add_subplot(234)
## --------------------plot intersection␣
↪lines-------------------------------------------------------------
for i in range (len(line_elements)):
    plot_line(ax, ob=line_elements4[i], color=RED, zorder=10, linewidth=1,␣
↪alpha=1)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)
ax.set_title('Grid Line 4 \n delta_x = 2, delta_y = 2 \n nl = 150')



ax = fig.add_subplot(235)
## --------------------plot intersection␣
↪lines-------------------------------------------------------------
for i in range (len(line_elements)):
    plot_line(ax, ob=line_elements5[i], color=RED, zorder=10, linewidth=1,␣
↪alpha=1)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)
ax.set_title('Grid Line 5 \n delta_x = 1, delta_y = 1000 \n nl = 200)')



ax = fig.add_subplot(236)
## --------------------plot intersection␣
↪lines-------------------------------------------------------------
for i in range (len(line_elements)):
    plot_line(ax, ob=line_elements6[i], color=RED, zorder=10, linewidth=1,␣
↪alpha=1)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)
ax.set_title('Grid Line 6 \n delta_x = 2, delta_y = 0.5 \n nl = 150')
```

```
[131]: Text(0.5, 1.0, 'Grid Line 6 \n delta_x = 2, delta_y = 0.5 \n nl = 150')
```



5.8.1 Zig-zag the lines

MATALB CODE:

```
[xc, yc] = poly2cw(xc, yc); [xb, yb] = bufferm2('xy', xc, yc, 0.4, 'in');

seg = zeros(0,2); dirr = true;

for ii = 1:nl [isin, inseg] = lineinpolygon(xl(1,ii), yl(1,ii), ... xl(2,ii),
yl(2,ii), xb, yb); if isin if dirr seg = [seg; NaN,NaN;inseg(1:end-1,:)]; else
seg = [seg; NaN,NaN;inseg(end-1:-1:1,:)]; end dirr =  dirr; end end
```

[132]:
```
line_segment_in, line_segment_out = LineInPolygonSegmentation(line_elements[2],␣
↪polygon_buffered)
```

```
MULTIPOINT (-10.0201 -20.1, -9.979900000000001 20.1)
MULTILINESTRING ((-9.979900000000001 20.1, -10.0201 -20.1))


GEOMETRYCOLLECTION EMPTY
```

[133]:
```
list_lines = []
for line in line_segment_in.geoms:
    list_lines.append(line)

list_line_reversed = list_lines[::-1]
reversed_segment = MultiLineString(tuple(list_line_reversed))
reversed_segment
```

[133]:

[134]:
```
def reverse_MultiLineString(multilinestring):
    '''
    This function reverse the Shapely MultiLineString Object

    Input/argument:
        - multilinestring: a shapely MultiLineString object

    Return : the MultiLineString object reversed
    '''

    list_lines = []
    for line in multilinestring.geoms:
        list_lines.append(line)

    list_line_reversed = list_lines[::-1]
    reversed_multilinestring = MultiLineString(tuple(list_line_reversed))
    return reversed_multilinestring
```

[135]:
```
segment_inside_collection_test = [] # empty list
line_segment_in = reverse_MultiLineString(line_segment_in)
segment_inside_collection_test.append (line_segment_in)

line_segment_in2, line_segment_out2 =␣
 ↪LineInPolygonSegmentation(line_elements[3], polygon_buffered)
segment_inside_collection_test.append (line_segment_in2)
segment_inside_collection_test
```

```
MULTIPOINT (-9.020099999999999 -20.1, -8.979900000000001 20.1,
-9.019113964647921 -19.11396464792147, -9.013521187611603 -13.52118761160264,
-8.986561391480455 13.4386085195447, -8.981390348570521 18.60965142947945)
MULTILINESTRING ((-8.979900000000001 20.1, -8.981390348570521
18.60965142947945), (-8.986561391480455 13.4386085195447, -9.013521187611603
-13.52118761160264), (-9.019113964647921 -19.11396464792147, -9.020099999999999
-20.1))


MULTILINESTRING ((-8.981390348570521 18.60965142947945, -8.986561391480455
13.4386085195447), (-9.013521187611603 -13.52118761160264, -9.019113964647921
-19.11396464792147))
```

[135]:
```
[<shapely.geometry.multilinestring.MultiLineString at 0x7f8530338898>,
 <shapely.geometry.multilinestring.MultiLineString at 0x7f852b9455f8>]
```

[136]:
```
# segment_inside_collection_test =␣
 ↪MultiLineString(tuple(segment_inside_collection_test))
```

- polygon_buffered --- the output of bufferm2 function (shapely polygon.buffer function equivalent)
- xl, yl -- created lines

[137]:
```
# line_elements = grid_line_creation(polygon, delta_x = 1, delta_y = 1000, nl =␣
 ↪150)

dirr = True
segment_inside_collection = [] # empty list

# from 0 to nl-1
for i in range (nl):
#     intersection_points2  = LinePolygonIntersectionPoints(line_example2,␣
 ↪polygon_buffered)
    line_segment_in, line_segment_out =␣
 ↪LineInPolygonSegmentation(line_elements[i], polygon_buffered)
```

```
    # if there is intersection between polygon and line
    if (line_segment_in is not None):
        if (dirr):
            segment_inside_collection.append (line_segment_in)
        else:
            segment_inside_collection.append␣
↪(reverse_MultiLineString(line_segment_in))
        dirr = ~dirr
```

```
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
MULTIPOINT (-10.0201 -20.1, -9.979900000000001 20.1)
MULTILINESTRING ((-9.979900000000001 20.1, -10.0201 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (-9.020099999999999 -20.1, -8.979900000000001 20.1,
-9.019113964647921 -19.11396464792147, -9.013521187611603 -13.52118761160264,
-8.986561391480455 13.4386085195447, -8.981390348570521 18.60965142947945)
MULTILINESTRING ((-8.979900000000001 20.1, -8.981390348570521
18.60965142947945), (-8.986561391480455 13.4386085195447, -9.013521187611603
-13.52118761160264), (-9.019113964647921 -19.11396464792147, -9.020099999999999
-20.1))


MULTILINESTRING ((-8.981390348570521 18.60965142947945, -8.986561391480455
13.4386085195447), (-9.013521187611603 -13.52118761160264, -9.019113964647921
-19.11396464792147))


MULTIPOINT (-8.020099999999999 -20.1, -7.9799 20.1, -8.019529381149237
-19.52938114923742, -7.992274150577748 7.725849422251899, -7.989308978569995
10.69102143000523, -7.980616500327026 19.38349967297389)
MULTILINESTRING ((-7.9799 20.1, -7.980616500327026 19.38349967297389),
(-7.989308978569995 10.69102143000523, -7.992274150577748 7.725849422251899),
(-8.019529381149237 -19.52938114923742, -8.020099999999999 -20.1))


MULTILINESTRING ((-7.980616500327026 19.38349967297389, -7.989308978569995
10.69102143000523), (-7.992274150577748 7.725849422251899, -8.019529381149237
-19.52938114923742))


MULTIPOINT (-7.0201 -20.1, -6.9799 20.1, -7.019729135406741 -19.72913540674106,
-6.991260151758692 8.739848241308067, -6.989076391217061 10.92360878293858,
-6.980678719062524 19.32128093747534)
MULTILINESTRING ((-6.9799 20.1, -6.980678719062524 19.32128093747534),
(-6.989076391217061 10.92360878293858, -6.991260151758692 8.739848241308067),
(-7.019729135406741 -19.72913540674106, -7.0201 -20.1))


MULTILINESTRING ((-6.980678719062524 19.32128093747534, -6.989076391217061
10.92360878293858), (-6.991260151758692 8.739848241308067, -7.019729135406741
-19.72913540674106))


MULTIPOINT (-6.0201 -20.1, -5.9799 20.1, -6.019448150577321 -19.44815057732056,
-6.01201943288986 -12.01943288985974, -6.011716450362085 -11.71645036208482,
-5.992753197878065 7.246802121935357, -5.987735027908111 12.26497209188901,
-5.980816156323892 19.18384367610732)
MULTILINESTRING ((-5.9799 20.1, -5.980816156323892 19.18384367610732),
(-5.987735027908111 12.26497209188901, -5.992753197878065 7.246802121935357),
(-6.011716450362085 -11.71645036208482, -6.01201943288986 -12.01943288985974),
(-6.019448150577321 -19.44815057732056, -6.0201 -20.1))


MULTILINESTRING ((-5.980816156323892 19.18384367610732, -5.987735027908111
12.26497209188901), (-5.992753197878065 7.246802121935357, -6.011716450362085
-11.71645036208482), (-6.01201943288986 -12.01943288985974, -6.019448150577321
-19.44815057732056))


MULTIPOINT (-5.0201 -20.1, -4.9799 20.1, -5.019078298917039 -19.07829891703869,
-5.014514788606887 -14.51478860688721, -4.986208478326869 13.79152167313094,
-4.981120625437242 18.8793745627579)
MULTILINESTRING ((-4.9799 20.1, -4.981120625437242 18.8793745627579),
(-4.986208478326869 13.79152167313094, -5.014514788606887 -14.51478860688721),
(-5.019078298917039 -19.07829891703869, -5.0201 -20.1))


MULTILINESTRING ((-4.981120625437242 18.8793745627579, -4.986208478326869
13.79152167313094), (-5.014514788606887 -14.51478860688721, -5.019078298917039
-19.07829891703869))


MULTIPOINT (-4.0201 -20.1, -3.9799 20.1, -4.018635764375806 -18.6357643758062,
-4.015528894238819 -15.52889423881876, -3.985027051053943 14.97294894605701,
-3.981477615592526 18.52238440747418)
MULTILINESTRING ((-3.9799 20.1, -3.981477615592526 18.52238440747418),
(-3.985027051053943 14.97294894605701, -4.015528894238819 -15.52889423881876),
(-4.018635764375806 -18.6357643758062, -4.0201 -20.1))
```

```
MULTILINESTRING ((-3.981477615592526 18.52238440747418, -3.985027051053943
14.97294894605701), (-4.015528894238819 -15.52889423881876, -4.018635764375806
-18.6357643758062))


MULTIPOINT (-3.0201 -20.1, -2.9799 20.1, -3.018168845290127 -18.16884529012732,
-3.016023527041979 -16.02352704197887)
MULTILINESTRING ((-2.9799 20.1, -3.016023527041979 -16.02352704197887),
(-3.018168845290127 -18.16884529012732, -3.0201 -20.1))


MULTILINESTRING ((-3.016023527041979 -16.02352704197887, -3.018168845290127
-18.16884529012732))


MULTIPOINT (-2.0201 -20.1, -1.9799 20.1, -2.017701926204449 -17.70192620444843,
-2.0163983586302 -16.39835863019946)
MULTILINESTRING ((-1.9799 20.1, -2.0163983586302 -16.39835863019946),
(-2.017701926204449 -17.70192620444843, -2.0201 -20.1))


MULTILINESTRING ((-2.0163983586302 -16.39835863019946, -2.017701926204449
-17.70192620444843))


MULTIPOINT (-1.020100000000001 -20.1, -0.9798999999999998 20.1)
MULTILINESTRING ((-0.9798999999999998 20.1, -1.020100000000001 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (-0.02010000000000112 -20.1, 0.02010000000000023 20.1)
MULTILINESTRING ((0.02010000000000023 20.1, -0.02010000000000112 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (0.9798999999999989 -20.1, 1.020099999999999 20.1, 1.016352329542646
16.3523295426456, 1.017273287434802 17.2732874348017)
MULTILINESTRING ((1.020099999999999 20.1, 1.017273287434802 17.2732874348017),
(1.016352329542646 16.3523295426456, 0.9798999999999989 -20.1))


MULTILINESTRING ((1.017273287434802 17.2732874348017, 1.016352329542646
16.3523295426456))
```

```
MULTIPOINT (1.9799 -20.1, 2.020099999999999 20.1, 2.015890765244237
15.89076524423744, 2.018554090481566 18.55409048156628, 1.982234573473017
-17.7654265269829, 1.983938056357376 -16.06194364262405)
MULTILINESTRING ((2.020099999999999 20.1, 2.018554090481566 18.55409048156628),
(2.015890765244237 15.89076524423744, 1.983938056357376 -16.06194364262405),
(1.982234573473017 -17.7654265269829, 1.9799 -20.1))


MULTILINESTRING ((2.018554090481566 18.55409048156628, 2.015890765244237
15.89076524423744), (1.983938056357376 -16.06194364262405, 1.982234573473017
-17.7654265269829))


MULTIPOINT (2.9799 -20.1, 3.0201 20.1, 3.015385996418334 15.38599641833372,
3.018882130628277 18.88213062827696, 2.981847008686459 -18.15299131354091,
2.984969482251395 -15.03051774860548)
MULTILINESTRING ((3.0201 20.1, 3.018882130628277 18.88213062827696),
(3.015385996418334 15.38599641833372, 2.984969482251395 -15.03051774860548),
(2.981847008686459 -18.15299131354091, 2.9799 -20.1))


MULTILINESTRING ((3.018882130628277 18.88213062827696, 3.015385996418334
15.38599641833372), (2.984969482251395 -15.03051774860548, 2.981847008686459
-18.15299131354091))


MULTIPOINT (3.979899999999999 -20.1, 4.020099999999999 20.1, 4.014726255930643
14.72625593064327, 4.019210121088291 19.21012108829037, 3.981459443899901
-18.54055610009891, 3.986136941402819 -13.86305859718085)
MULTILINESTRING ((4.020099999999999 20.1, 4.019210121088291 19.21012108829037),
(4.014726255930643 14.72625593064327, 3.986136941402819 -13.86305859718085),
(3.981459443899901 -18.54055610009891, 3.979899999999999 -20.1))


MULTILINESTRING ((4.019210121088291 19.21012108829037, 4.014726255930643
14.72625593064327), (3.986136941402819 -13.86305859718085, 3.981459443899901
-18.54055610009891))


MULTIPOINT (4.979900000000001 -20.1, 5.020100000000001 20.1, 5.013156168802753
13.15616880275393, 5.019419241143639 19.4192411436394, 4.981245471822906
-18.75452817709423, 4.987549074829227 -12.45092517077374, 4.992067225834511
-7.932774165489135, 4.998454979696845 -1.545020303155085)
MULTILINESTRING ((5.020100000000001 20.1, 5.019419241143639 19.4192411436394),
(5.013156168802753 13.15616880275393, 4.998454979696845 -1.545020303155085),
(4.992067225834511 -7.932774165489135, 4.987549074829227 -12.45092517077374),
(4.981245471822906 -18.75452817709423, 4.979900000000001 -20.1))
```

```
MULTILINESTRING ((5.019419241143639 19.4192411436394, 5.013156168802753
13.15616880275393), (4.998454979696845 -1.545020303155085, 4.992067225834511
-7.932774165489135), (4.987549074829227 -12.45092517077374, 4.981245471822906
-18.75452817709423))


MULTIPOINT (5.979900000000001 -20.1, 6.020100000000001 20.1, 6.012234419039932
12.23441903993229, 6.019615066068246 19.61506606824588, 5.981104042494465
-18.89595750553508, 5.990091903627411 -9.90809637258959, 5.990596276333743
-9.403723666256608, 6.008644205780396 8.644205780395454)
MULTILINESTRING ((6.020100000000001 20.1, 6.019615066068246 19.61506606824588),
(6.012234419039932 12.23441903993229, 6.008644205780396 8.644205780395454),
(5.990596276333743 -9.403723666256608, 5.990091903627411 -9.90809637258959),
(5.981104042494465 -18.89595750553508, 5.979900000000001 -20.1))


MULTILINESTRING ((6.019615066068246 19.61506606824588, 6.012234419039932
12.23441903993229), (6.008644205780396 8.644205780395454, 5.990596276333743
-9.403723666256608), (5.990091903627411 -9.90809637258959, 5.981104042494465
-18.89595750553508))


MULTIPOINT (6.979900000000002 -20.1, 7.0201 20.1, 7.012823856216696
12.82385621669549, 7.019564490706952 19.56449070695215, 6.980989637910159
-19.01036208984158, 6.988920867061899 -11.07913293810089, 6.992008042771287
-7.991957228712335, 7.008542794934124 8.542794934124073)
MULTILINESTRING ((7.0201 20.1, 7.019564490706952 19.56449070695215),
(7.012823856216696 12.82385621669549, 7.008542794934124 8.542794934124073),
(6.992008042771287 -7.991957228712335, 6.988920867061899 -11.07913293810089),
(6.980989637910159 -19.01036208984158, 6.979900000000002 -20.1))


MULTILINESTRING ((7.019564490706952 19.56449070695215, 7.012823856216696
12.82385621669549), (7.008542794934124 8.542794934124073, 6.992008042771287
-7.991957228712335), (6.988920867061899 -11.07913293810089, 6.980989637910159
-19.01036208984158))


MULTIPOINT (7.979900000000002 -20.1, 8.020099999999999 20.1, 8.014115961272038
14.11596127203866, 8.0193895059494 19.38950594939988, 7.981405024758268
-18.59497524173239, 7.986831489012969 -13.16851098703038, 7.99683575105567
-3.164248944329993, 7.999887384646775 -0.1126153532250525)
MULTILINESTRING ((8.020099999999999 20.1, 8.0193895059494 19.38950594939988),
(8.014115961272038 14.11596127203866, 7.999887384646775 -0.1126153532250525),
(7.99683575105567 -3.164248944329993, 7.986831489012969 -13.16851098703038),
(7.981405024758268 -18.59497524173239, 7.979900000000002 -20.1))
```

```
MULTILINESTRING ((8.0193895059494 19.38950594939988, 8.014115961272038
14.11596127203866), (7.999887384646775 -0.1126153532250525, 7.99683575105567
-3.164248944329993), (7.986831489012969 -13.16851098703038, 7.981405024758268
-18.59497524173239))


MULTIPOINT (8.979900000000001 -20.1, 9.020099999999999 20.1, 9.01719107702873
17.19107702873, 9.018126780111102 18.12678011110124)
MULTILINESTRING ((9.020099999999999 20.1, 9.018126780111102 18.12678011110124),
(9.01719107702873 17.19107702873, 8.979900000000001 -20.1))


MULTILINESTRING ((9.018126780111102 18.12678011110124, 9.01719107702873
17.19107702873))


MULTIPOINT (9.979900000000001 -20.1, 10.0201 20.1)
MULTILINESTRING ((10.0201 20.1, 9.979900000000001 -20.1))


GEOMETRYCOLLECTION EMPTY


GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
```

5.9 Integrate the zig-zag line segmentation function

[138]:
```python
def zig_zag_segmentation(line_elements):
    '''
    Input
        - line_elements: a list object, contains a list of created grid lines␣
↪(each element is a LineString object)

    Output:
        - segment_inside_collection: a list object, each element is a␣
↪MultiLineString object containing the inside polygon portion of line
        - segment_outside_collection: a list, each element is a MultiLineStirng␣
↪object containing outside portion of the line
        - intersection_points_collection: list, each element is a MultiPoint␣
↪object containing the intersection point of linea and polygon
    '''

    # line_elements = grid_line_creation(polygon, delta_x = 1, delta_y = 1000,␣
↪nl = 150)

    dirr = True
    dirr_out = True
    segment_inside_collection = [] # empty list for the inside portion of the␣
↪lines
    segment_outside_collection = [] # empty list for the outside portion of the␣
↪lines
    intersection_points_collection = [] # empty list for all intersection points

    # from 0 to nl-1
    for i in range (nl):
    #     intersection_points2  = LinePolygonIntersectionPoints(line_example2,␣
↪polygon_buffered)
        intersection_points = LinePolygonIntersectionPoints(line_elements[i],␣
↪polygon_buffered)
        line_segment_in, line_segment_out =␣
↪LineInPolygonSegmentation(line_elements[i], polygon_buffered)
```

```
        if (len(intersection_points.geoms) != 0 ):
            intersection_points_collection.append(intersection_points)
#             if (dirr):
#                     segment_inside_collection.append (line_segment_in)
#             else:
#                 segment_inside_collection.append␣
↪(reverse_MultiLineString(line_segment_in))
#             dirr = ~dirr


        # if there is intersection between polygon and line
        if (line_segment_in is not None):
            if (dirr):
                segment_inside_collection.append (line_segment_in)
            else:
                segment_inside_collection.append␣
↪(reverse_MultiLineString(line_segment_in))
            dirr = ~dirr

        if (line_segment_out is not None):
            if (dirr_out):
                segment_outside_collection.append (line_segment_out)
            else:
                segment_outside_collection.append␣
↪(reverse_MultiLineString(line_segment_out))
            dirr_out = ~dirr_out



    return segment_inside_collection, segment_outside_collection,␣
↪intersection_points_collection
```

[139]:
```
segment_inside_collection, segment_out_collection,␣
↪intersection_points_collection = zig_zag_segmentation(line_elements)
```

```
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
MULTIPOINT (-10.0201 -20.1, -9.979900000000001 20.1)
MULTIPOINT (-10.0201 -20.1, -9.979900000000001 20.1)
MULTILINESTRING ((-9.979900000000001 20.1, -10.0201 -20.1))


GEOMETRYCOLLECTION EMPTY
```

```
MULTIPOINT (-9.020099999999999 -20.1, -8.979900000000001 20.1,
-9.019113964647921 -19.11396464792147, -9.013521187611603 -13.52118761160264,
-8.986561391480455 13.4386085195447, -8.981390348570521 18.60965142947945)
MULTIPOINT (-9.020099999999999 -20.1, -8.979900000000001 20.1,
-9.019113964647921 -19.11396464792147, -9.013521187611603 -13.52118761160264,
-8.986561391480455 13.4386085195447, -8.981390348570521 18.60965142947945)
MULTILINESTRING ((-8.979900000000001 20.1, -8.981390348570521
18.60965142947945), (-8.986561391480455 13.4386085195447, -9.013521187611603
-13.52118761160264), (-9.019113964647921 -19.11396464792147, -9.020099999999999
-20.1))


MULTILINESTRING ((-8.981390348570521 18.60965142947945, -8.986561391480455
13.4386085195447), (-9.013521187611603 -13.52118761160264, -9.019113964647921
-19.11396464792147))


MULTIPOINT (-8.020099999999999 -20.1, -7.9799 20.1, -8.019529381149237
-19.52938114923742, -7.992274150577748 7.725849422251899, -7.989308978569995
10.69102143000523, -7.980616500327026 19.38349967297389)
MULTIPOINT (-8.020099999999999 -20.1, -7.9799 20.1, -8.019529381149237
-19.52938114923742, -7.992274150577748 7.725849422251899, -7.989308978569995
10.69102143000523, -7.980616500327026 19.38349967297389)
MULTILINESTRING ((-7.9799 20.1, -7.980616500327026 19.38349967297389),
(-7.989308978569995 10.69102143000523, -7.992274150577748 7.725849422251899),
(-8.019529381149237 -19.52938114923742, -8.020099999999999 -20.1))


MULTILINESTRING ((-7.980616500327026 19.38349967297389, -7.989308978569995
10.69102143000523), (-7.992274150577748 7.725849422251899, -8.019529381149237
-19.52938114923742))


MULTIPOINT (-7.0201 -20.1, -6.9799 20.1, -7.019729135406741 -19.72913540674106,
-6.991260151758692 8.739848241308067, -6.989076391217061 10.92360878293858,
-6.980678719062524 19.32128093747534)
MULTIPOINT (-7.0201 -20.1, -6.9799 20.1, -7.019729135406741 -19.72913540674106,
-6.991260151758692 8.739848241308067, -6.989076391217061 10.92360878293858,
-6.980678719062524 19.32128093747534)
MULTILINESTRING ((-6.9799 20.1, -6.980678719062524 19.32128093747534),
(-6.989076391217061 10.92360878293858, -6.991260151758692 8.739848241308067),
(-7.019729135406741 -19.72913540674106, -7.0201 -20.1))


MULTILINESTRING ((-6.980678719062524 19.32128093747534, -6.989076391217061
10.92360878293858), (-6.991260151758692 8.739848241308067, -7.019729135406741
-19.72913540674106))
```

```
MULTIPOINT (-6.0201 -20.1, -5.9799 20.1, -6.019448150577321 -19.44815057732056,
-6.01201943288986 -12.01943288985974, -6.011716450362085 -11.71645036208482,
-5.992753197878065 7.246802121935357, -5.987735027908111 12.26497209188901,
-5.980816156323892 19.18384367610732)
MULTIPOINT (-6.0201 -20.1, -5.9799 20.1, -6.019448150577321 -19.44815057732056,
-6.01201943288986 -12.01943288985974, -6.011716450362085 -11.71645036208482,
-5.992753197878065 7.246802121935357, -5.987735027908111 12.26497209188901,
-5.980816156323892 19.18384367610732)
MULTILINESTRING ((-5.9799 20.1, -5.980816156323892 19.18384367610732),
(-5.987735027908111 12.26497209188901, -5.992753197878065 7.246802121935357),
(-6.011716450362085 -11.71645036208482, -6.01201943288986 -12.01943288985974),
(-6.019448150577321 -19.44815057732056, -6.0201 -20.1))


MULTILINESTRING ((-5.980816156323892 19.18384367610732, -5.987735027908111
12.26497209188901), (-5.992753197878065 7.246802121935357, -6.011716450362085
-11.71645036208482), (-6.01201943288986 -12.01943288985974, -6.019448150577321
-19.44815057732056))


MULTIPOINT (-5.0201 -20.1, -4.9799 20.1, -5.019078298917039 -19.07829891703869,
-5.014514788606887 -14.51478860688721, -4.986208478326869 13.79152167313094,
-4.981120625437242 18.8793745627579)
MULTIPOINT (-5.0201 -20.1, -4.9799 20.1, -5.019078298917039 -19.07829891703869,
-5.014514788606887 -14.51478860688721, -4.986208478326869 13.79152167313094,
-4.981120625437242 18.8793745627579)
MULTILINESTRING ((-4.9799 20.1, -4.981120625437242 18.8793745627579),
(-4.986208478326869 13.79152167313094, -5.014514788606887 -14.51478860688721),
(-5.019078298917039 -19.07829891703869, -5.0201 -20.1))


MULTILINESTRING ((-4.981120625437242 18.8793745627579, -4.986208478326869
13.79152167313094), (-5.014514788606887 -14.51478860688721, -5.019078298917039
-19.07829891703869))


MULTIPOINT (-4.0201 -20.1, -3.9799 20.1, -4.018635764375806 -18.6357643758062,
-4.015528894238819 -15.52889423881876, -3.985027051053943 14.97294894605701,
-3.981477615592526 18.52238440747418)
MULTIPOINT (-4.0201 -20.1, -3.9799 20.1, -4.018635764375806 -18.6357643758062,
-4.015528894238819 -15.52889423881876, -3.985027051053943 14.97294894605701,
-3.981477615592526 18.52238440747418)
MULTILINESTRING ((-3.9799 20.1, -3.981477615592526 18.52238440747418),
(-3.985027051053943 14.97294894605701, -4.015528894238819 -15.52889423881876),
(-4.018635764375806 -18.6357643758062, -4.0201 -20.1))
```

```
MULTILINESTRING ((-3.981477615592526 18.52238440747418, -3.985027051053943
14.97294894605701), (-4.015528894238819 -15.52889423881876, -4.018635764375806
-18.6357643758062))


MULTIPOINT (-3.0201 -20.1, -2.9799 20.1, -3.018168845290127 -18.16884529012732,
-3.016023527041979 -16.02352704197887)
MULTIPOINT (-3.0201 -20.1, -2.9799 20.1, -3.018168845290127 -18.16884529012732,
-3.016023527041979 -16.02352704197887)
MULTILINESTRING ((-2.9799 20.1, -3.016023527041979 -16.02352704197887),
(-3.018168845290127 -18.16884529012732, -3.0201 -20.1))


MULTILINESTRING ((-3.016023527041979 -16.02352704197887, -3.018168845290127
-18.16884529012732))


MULTIPOINT (-2.0201 -20.1, -1.9799 20.1, -2.017701926204449 -17.70192620444843,
-2.0163983586302 -16.39835863019946)
MULTIPOINT (-2.0201 -20.1, -1.9799 20.1, -2.017701926204449 -17.70192620444843,
-2.0163983586302 -16.39835863019946)
MULTILINESTRING ((-1.9799 20.1, -2.0163983586302 -16.39835863019946),
(-2.017701926204449 -17.70192620444843, -2.0201 -20.1))


MULTILINESTRING ((-2.0163983586302 -16.39835863019946, -2.017701926204449
-17.70192620444843))


MULTIPOINT (-1.020100000000001 -20.1, -0.9798999999999998 20.1)
MULTIPOINT (-1.020100000000001 -20.1, -0.9798999999999998 20.1)
MULTILINESTRING ((-0.9798999999999998 20.1, -1.020100000000001 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (-0.02010000000000112 -20.1, 0.02010000000000023 20.1)
MULTIPOINT (-0.02010000000000112 -20.1, 0.02010000000000023 20.1)
MULTILINESTRING ((0.02010000000000023 20.1, -0.02010000000000112 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (0.9798999999999989 -20.1, 1.020099999999999 20.1, 1.016352329542646
16.3523295426456, 1.017273287434802 17.2732874348017)
MULTIPOINT (0.9798999999999989 -20.1, 1.020099999999999 20.1, 1.016352329542646
16.3523295426456, 1.017273287434802 17.2732874348017)
MULTILINESTRING ((1.020099999999999 20.1, 1.017273287434802 17.2732874348017),
(1.016352329542646 16.3523295426456, 0.9798999999999989 -20.1))


MULTILINESTRING ((1.017273287434802 17.2732874348017, 1.016352329542646
16.3523295426456))


MULTIPOINT (1.9799 -20.1, 2.020099999999999 20.1, 2.015890765244237
15.89076524423744, 2.018554090481566 18.55409048156628, 1.982234573473017
-17.7654265269829, 1.983938056357376 -16.06194364262405)
MULTIPOINT (1.9799 -20.1, 2.020099999999999 20.1, 2.015890765244237
15.89076524423744, 2.018554090481566 18.55409048156628, 1.982234573473017
-17.7654265269829, 1.983938056357376 -16.06194364262405)
MULTILINESTRING ((2.020099999999999 20.1, 2.018554090481566 18.55409048156628),
(2.015890765244237 15.89076524423744, 1.983938056357376 -16.06194364262405),
(1.982234573473017 -17.7654265269829, 1.9799 -20.1))


MULTILINESTRING ((2.018554090481566 18.55409048156628, 2.015890765244237
15.89076524423744), (1.983938056357376 -16.06194364262405, 1.982234573473017
-17.7654265269829))


MULTIPOINT (2.9799 -20.1, 3.0201 20.1, 3.015385996418334 15.38599641833372,
3.018882130628277 18.88213062827696, 2.981847008686459 -18.15299131354091,
2.984969482251395 -15.03051774860548)
MULTIPOINT (2.9799 -20.1, 3.0201 20.1, 3.015385996418334 15.38599641833372,
3.018882130628277 18.88213062827696, 2.981847008686459 -18.15299131354091,
2.984969482251395 -15.03051774860548)
MULTILINESTRING ((3.0201 20.1, 3.018882130628277 18.88213062827696),
(3.015385996418334 15.38599641833372, 2.984969482251395 -15.03051774860548),
(2.981847008686459 -18.15299131354091, 2.9799 -20.1))


MULTILINESTRING ((3.018882130628277 18.88213062827696, 3.015385996418334
15.38599641833372), (2.984969482251395 -15.03051774860548, 2.981847008686459
-18.15299131354091))


MULTIPOINT (3.979899999999999 -20.1, 4.020099999999999 20.1, 4.014726255930643
14.72625593064327, 4.019210121088291 19.21012108829037, 3.981459443899901
-18.54055610009891, 3.986136941402819 -13.86305859718085)
MULTIPOINT (3.979899999999999 -20.1, 4.020099999999999 20.1, 4.014726255930643
14.72625593064327, 4.019210121088291 19.21012108829037, 3.981459443899901
-18.54055610009891, 3.986136941402819 -13.86305859718085)
MULTILINESTRING ((4.020099999999999 20.1, 4.019210121088291 19.21012108829037),
(4.014726255930643 14.72625593064327, 3.986136941402819 -13.86305859718085),
(3.981459443899901 -18.54055610009891, 3.979899999999999 -20.1))


MULTILINESTRING ((4.019210121088291 19.21012108829037, 4.014726255930643
14.72625593064327), (3.986136941402819 -13.86305859718085, 3.981459443899901
-18.54055610009891))


MULTIPOINT (4.979900000000001 -20.1, 5.020100000000001 20.1, 5.013156168802753
13.15616880275393, 5.019419241143639 19.4192411436394, 4.981245471822906
-18.75452817709423, 4.987549074829227 -12.45092517077374, 4.992067225834511
-7.932774165489135, 4.998454979696845 -1.545020303155085)
MULTIPOINT (4.979900000000001 -20.1, 5.020100000000001 20.1, 5.013156168802753
13.15616880275393, 5.019419241143639 19.4192411436394, 4.981245471822906
-18.75452817709423, 4.987549074829227 -12.45092517077374, 4.992067225834511
-7.932774165489135, 4.998454979696845 -1.545020303155085)
MULTILINESTRING ((5.020100000000001 20.1, 5.019419241143639 19.4192411436394),
(5.013156168802753 13.15616880275393, 4.998454979696845 -1.545020303155085),
(4.992067225834511 -7.932774165489135, 4.987549074829227 -12.45092517077374),
(4.981245471822906 -18.75452817709423, 4.979900000000001 -20.1))


MULTILINESTRING ((5.019419241143639 19.4192411436394, 5.013156168802753
13.15616880275393), (4.998454979696845 -1.545020303155085, 4.992067225834511
-7.932774165489135), (4.987549074829227 -12.45092517077374, 4.981245471822906
-18.75452817709423))


MULTIPOINT (5.979900000000001 -20.1, 6.020100000000001 20.1, 6.012234419039932
12.23441903993229, 6.019615066068246 19.61506606824588, 5.981104042494465
-18.89595750553508, 5.990091903627411 -9.90809637258959, 5.990596276333743
-9.403723666256608, 6.008644205780396 8.644205780395454)
MULTIPOINT (5.979900000000001 -20.1, 6.020100000000001 20.1, 6.012234419039932
12.23441903993229, 6.019615066068246 19.61506606824588, 5.981104042494465
-18.89595750553508, 5.990091903627411 -9.90809637258959, 5.990596276333743
-9.403723666256608, 6.008644205780396 8.644205780395454)
MULTILINESTRING ((6.020100000000001 20.1, 6.019615066068246 19.61506606824588),
(6.012234419039932 12.23441903993229, 6.008644205780396 8.644205780395454),
(5.990596276333743 -9.403723666256608, 5.990091903627411 -9.90809637258959),
(5.981104042494465 -18.89595750553508, 5.979900000000001 -20.1))


MULTILINESTRING ((6.019615066068246 19.61506606824588, 6.012234419039932
12.23441903993229), (6.008644205780396 8.644205780395454, 5.990596276333743
-9.403723666256608), (5.990091903627411 -9.90809637258959, 5.981104042494465
-18.89595750553508))
```

```
MULTIPOINT (6.979900000000002 -20.1, 7.0201 20.1, 7.012823856216696
12.82385621669549, 7.019564490706952 19.56449070695215, 6.980989637910159
-19.01036208984158, 6.988920867061899 -11.07913293810089, 6.992008042771287
-7.991957228712335, 7.008542794934124 8.542794934124073)
MULTIPOINT (6.979900000000002 -20.1, 7.0201 20.1, 7.012823856216696
12.82385621669549, 7.019564490706952 19.56449070695215, 6.980989637910159
-19.01036208984158, 6.988920867061899 -11.07913293810089, 6.992008042771287
-7.991957228712335, 7.008542794934124 8.542794934124073)
MULTILINESTRING ((7.0201 20.1, 7.019564490706952 19.56449070695215),
(7.012823856216696 12.82385621669549, 7.008542794934124 8.542794934124073),
(6.992008042771287 -7.991957228712335, 6.988920867061899 -11.07913293810089),
(6.980989637910159 -19.01036208984158, 6.979900000000002 -20.1))


MULTILINESTRING ((7.019564490706952 19.56449070695215, 7.012823856216696
12.82385621669549), (7.008542794934124 8.542794934124073, 6.992008042771287
-7.991957228712335), (6.988920867061899 -11.07913293810089, 6.980989637910159
-19.01036208984158))


MULTIPOINT (7.979900000000002 -20.1, 8.020099999999999 20.1, 8.014115961272038
14.11596127203866, 8.0193895059494 19.38950594939988, 7.981405024758268
-18.59497524173239, 7.986831489012969 -13.16851098703038, 7.99683575105567
-3.164248944329993, 7.999887384646775 -0.1126153532250525)
MULTIPOINT (7.979900000000002 -20.1, 8.020099999999999 20.1, 8.014115961272038
14.11596127203866, 8.0193895059494 19.38950594939988, 7.981405024758268
-18.59497524173239, 7.986831489012969 -13.16851098703038, 7.99683575105567
-3.164248944329993, 7.999887384646775 -0.1126153532250525)
MULTILINESTRING ((8.020099999999999 20.1, 8.0193895059494 19.38950594939988),
(8.014115961272038 14.11596127203866, 7.999887384646775 -0.1126153532250525),
(7.99683575105567 -3.164248944329993, 7.986831489012969 -13.16851098703038),
(7.981405024758268 -18.59497524173239, 7.979900000000002 -20.1))


MULTILINESTRING ((8.0193895059494 19.38950594939988, 8.014115961272038
14.11596127203866), (7.999887384646775 -0.1126153532250525, 7.99683575105567
-3.164248944329993), (7.986831489012969 -13.16851098703038, 7.981405024758268
-18.59497524173239))


MULTIPOINT (8.979900000000001 -20.1, 9.020099999999999 20.1, 9.01719107702873
17.19107702873, 9.018126780111102 18.12678011110124)
MULTIPOINT (8.979900000000001 -20.1, 9.020099999999999 20.1, 9.01719107702873
17.19107702873, 9.018126780111102 18.12678011110124)
MULTILINESTRING ((9.020099999999999 20.1, 9.018126780111102 18.12678011110124),
(9.01719107702873 17.19107702873, 8.979900000000001 -20.1))
```

```
MULTILINESTRING ((9.018126780111102 18.12678011110124, 9.01719107702873
17.19107702873))


MULTIPOINT (9.979900000000001 -20.1, 10.0201 20.1)
MULTIPOINT (9.979900000000001 -20.1, 10.0201 20.1)
MULTILINESTRING ((10.0201 20.1, 9.979900000000001 -20.1))


GEOMETRYCOLLECTION EMPTY


GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
```

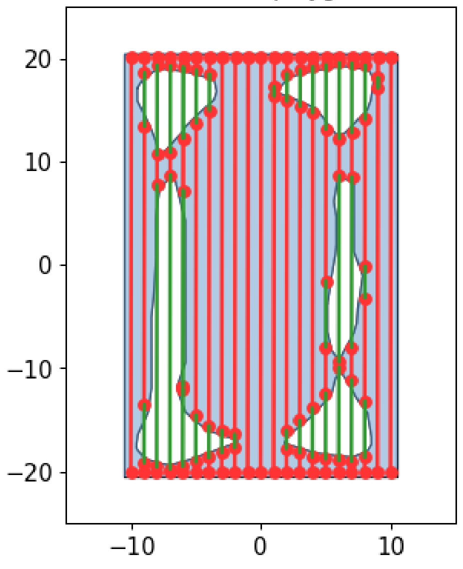
5.10 Visualize the final results

[140]:
```
### visualization of the line polygon segmentation

fig = pyplot.figure(1, figsize=SIZE, dpi=90)
ax = fig.add_subplot(121)

#----------------------plot polygon with␣
↪boundaries----------------------------------------------------------
plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=1, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
↪alpha=1)
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
↪alpha=1)
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
↪alpha=1)
```

```
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)

# segment_inside_collection, segment_out_collection,␣
 ↪intersection_points_collection
#----------------------plot the intersection␣
 ↪point----------------------------------------------------------
for intersection_points in intersection_points_collection:
    xs = [point.x for point in intersection_points.geoms]
    ys = [point.y for point in intersection_points.geoms]
    ax.plot(xs, ys, 'o', color=RED, zorder=10, alpha=1)
# ax.plot(intersection_points, 'o', color=RED, zorder=1, alpha=1)


## --------------------plot segmented␣
 ↪lines-------------------------------------------------------------
for in_seg in segment_inside_collection:
    for i in range (len(in_seg.geoms)):
        plot_line(ax, ob=in_seg.geoms[i], color=RED, zorder=10, linewidth=2,␣
 ↪alpha=1)


for out_seg in segment_out_collection:
    for i in range (len(out_seg.geoms)):
        plot_line(ax, ob=out_seg.geoms[i], color=GREEN, zorder=10, linewidth=2,␣
 ↪alpha=1)



patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Red Line - Inside polygon, Green Line -- Outside polygon/in Holes.
 ↪ Red dots - intersection point')
```

[140]:
```
Text(0.5, 1.0, 'Red Line - Inside polygon, Green Line -- Outside polygon/in
Holes. Red dots - intersection point')
```

Red Line - Inside polygon, Green Line -- Outside polygon/in Holes. Red dots - intersection point



6 Visaulization - Example 2 (change parameters dx)

[141]:
```
segment_inside_collection, segment_out_collection,␣
↪intersection_points_collection = zig_zag_segmentation(line_elements2)
```

```
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
MULTIPOINT (-10.0402 -20.1, -9.9598 20.1)
MULTIPOINT (-10.0402 -20.1, -9.9598 20.1)
MULTILINESTRING ((-9.9598 20.1, -10.0402 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (-8.0402 -20.1, -7.9598 20.1, -8.039043671316268 -19.52183565813426,
-7.984521804552391 7.739097723804414, -7.978629162784162 10.68541860791932,
-7.961221650374229 19.38917481288555)
MULTIPOINT (-8.0402 -20.1, -7.9598 20.1, -8.039043671316268 -19.52183565813426,
-7.984521804552391 7.739097723804414, -7.978629162784162 10.68541860791932,
-7.961221650374229 19.38917481288555)
MULTILINESTRING ((-7.9598 20.1, -7.961221650374229 19.38917481288555),
(-7.978629162784162 10.68541860791932, -7.984521804552391 7.739097723804414),
(-8.039043671316268 -19.52183565813426, -8.0402 -20.1))


MULTILINESTRING ((-7.961221650374229 19.38917481288555, -7.978629162784162
10.68541860791932), (-7.984521804552391 7.739097723804414, -8.039043671316268
-19.52183565813426))


MULTIPOINT (-6.0402 -20.1, -5.9598 20.1, -6.038910692338805 -19.45534616940267,
-6.023951864120448 -11.97593206022409, -6.023520931928925 -11.76046596446218,
-5.985580330699913 7.209834650043319, -5.975437108054204 12.28144597289791,
-5.961637585073032 19.18120746348381)
```

```
MULTIPOINT (-6.0402 -20.1, -5.9598 20.1, -6.038910692338805 -19.45534616940267,
-6.023951864120448 -11.97593206022409, -6.023520931928925 -11.76046596446218,
-5.985580330699913 7.209834650043319, -5.975437108054204 12.28144597289791,
-5.961637585073032 19.18120746348381)
```

```
MULTILINESTRING ((-5.9598 20.1, -5.961637585073032 19.18120746348381),
(-5.975437108054204 12.28144597289791, -5.985580330699913 7.209834650043319),
(-6.023520931928925 -11.76046596446218, -6.023951864120448 -11.97593206022409),
(-6.038910692338805 -19.45534616940267, -6.0402 -20.1))


MULTILINESTRING ((-5.961637585073032 19.18120746348381, -5.975437108054204
12.28144597289791), (-5.985580330699913 7.209834650043319, -6.023520931928925
-11.76046596446218), (-6.023951864120448 -11.97593206022409, -6.038910692338805
-19.45534616940267))


MULTIPOINT (-4.0402 -20.1, -3.9598 20.1, -4.037288939669229 -18.64446983461435,
-4.031026324507216 -15.51316225360831, -3.970018681360367 14.99065931981658,
-3.96296845108345 18.51577445827512)
```

```
MULTIPOINT (-4.0402 -20.1, -3.9598 20.1, -4.037288939669229 -18.64446983461435,
-4.031026324507216 -15.51316225360831, -3.970018681360367 14.99065931981658,
-3.96296845108345 18.51577445827512)
```

```
MULTILINESTRING ((-3.9598 20.1, -3.96296845108345 18.51577445827512),
(-3.970018681360367 14.99065931981658, -4.031026324507216 -15.51316225360831),
(-4.037288939669229 -18.64446983461435, -4.0402 -20.1))


MULTILINESTRING ((-3.96296845108345 18.51577445827512, -3.970018681360367
14.99065931981658), (-4.031026324507216 -15.51316225360831, -4.037288939669229
-18.64446983461435))


MULTIPOINT (-2.0402 -20.1, -1.9598 20.1, -2.035420390865414 -17.71019543270707,
-2.03278442862095 -16.39221431047501)
```

```
MULTIPOINT (-2.0402 -20.1, -1.9598 20.1, -2.035420390865414 -17.71019543270707,
-2.03278442862095 -16.39221431047501)
```

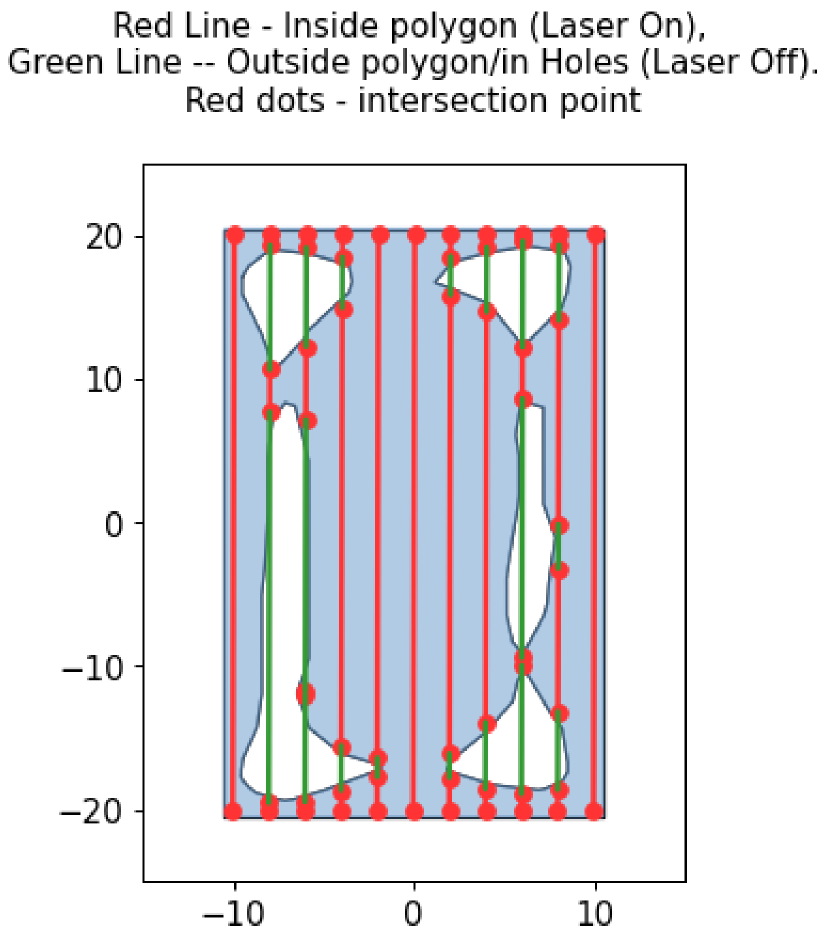
```
MULTILINESTRING ((-1.9598 20.1, -2.03278442862095 -16.39221431047501),
(-2.035420390865414 -17.71019543270707, -2.0402 -20.1))


MULTILINESTRING ((-2.03278442862095 -16.39221431047501, -2.035420390865414
-17.71019543270707))


MULTIPOINT (-0.04020000000000046 -20.1, 0.04020000000000046 20.1)
MULTIPOINT (-0.04020000000000046 -20.1, 0.04020000000000046 20.1)
MULTILINESTRING ((0.04020000000000046 20.1, -0.04020000000000046 -20.1))


GEOMETRYCOLLECTION EMPTY


MULTIPOINT (1.9598 -20.1, 2.0402 20.1, 2.031765496256225 15.88274812811269,
2.0371204554918 18.56022774589988, 1.964482912118618 -17.75854394069094,
1.967842945095648 -16.07852745217592)
MULTIPOINT (1.9598 -20.1, 2.0402 20.1, 2.031765496256225 15.88274812811269,
2.0371204554918 18.56022774589988, 1.964482912118618 -17.75854394069094,
1.967842945095648 -16.07852745217592)
MULTILINESTRING ((2.0402 20.1, 2.0371204554918 18.56022774589988),
(2.031765496256225 15.88274812811269, 1.967842945095648 -16.07852745217592),
(1.964482912118618 -17.75854394069094, 1.9598 -20.1))


MULTILINESTRING ((2.0371204554918 18.56022774589988, 2.031765496256225
15.88274812811269), (1.967842945095648 -16.07852745217592, 1.964482912118618
-17.75854394069094))


MULTIPOINT (3.9598 -20.1, 4.0402 20.1, 4.029406341343265 14.70317067163258,
4.038432847784005 19.21642389200252, 3.962933253565472 -18.53337321726377,
3.97223467446135 -13.88266276932495)
MULTIPOINT (3.9598 -20.1, 4.0402 20.1, 4.029406341343265 14.70317067163258,
4.038432847784005 19.21642389200252, 3.962933253565472 -18.53337321726377,
3.97223467446135 -13.88266276932495)
MULTILINESTRING ((4.0402 20.1, 4.038432847784005 19.21642389200252),
(4.029406341343265 14.70317067163258, 3.97223467446135 -13.88266276932495),
(3.962933253565472 -18.53337321726377, 3.9598 -20.1))


MULTILINESTRING ((4.038432847784005 19.21642389200252, 4.029406341343265
14.70317067163258), (3.97223467446135 -13.88266276932495, 3.962933253565472
-18.53337321726377))
```

```
MULTIPOINT (5.9598 -20.1, 6.0402 20.1, 6.024468447868689 12.23422393434466,
6.03923736105786 19.61868052892999, 5.962213220746366 -18.89338962681668,
5.980176684283778 -9.911657858110839, 5.981199248901871 -9.400375549064785,
6.01729280304709 8.646401523544617)
MULTIPOINT (5.9598 -20.1, 6.0402 20.1, 6.024468447868689 12.23422393434466,
6.03923736105786 19.61868052892999, 5.962213220746366 -18.89338962681668,
5.980176684283778 -9.911657858110839, 5.981199248901871 -9.400375549064785,
6.01729280304709 8.646401523544617)
MULTILINESTRING ((6.0402 20.1, 6.03923736105786 19.61868052892999),
(6.024468447868689 12.23422393434466, 6.01729280304709 8.646401523544617),
(5.981199248901871 -9.400375549064785, 5.980176684283778 -9.911657858110839),
(5.962213220746366 -18.89338962681668, 5.9598 -20.1))


MULTILINESTRING ((6.03923736105786 19.61868052892999, 6.024468447868689
12.23422393434466), (6.01729280304709 8.646401523544617, 5.981199248901871
-9.400375549064785), (5.980176684283778 -9.911657858110839, 5.962213220746366
-18.89338962681668))


MULTIPOINT (7.959800000000002 -20.1, 8.040200000000002 20.1, 8.028268448349095
14.13422417454794, 8.038772227349996 19.38611367499857, 7.96278962994533
-18.60518502733526, 7.973707049307949 -13.14647534602538, 7.99361407393468
-3.192963032659963, 7.999775582785051 -0.1122086074745767)
MULTIPOINT (7.959800000000002 -20.1, 8.040200000000002 20.1, 8.028268448349095
14.13422417454794, 8.038772227349996 19.38611367499857, 7.96278962994533
-18.60518502733526, 7.973707049307949 -13.14647534602538, 7.99361407393468
-3.192963032659963, 7.999775582785051 -0.1122086074745767)
MULTILINESTRING ((8.040200000000002 20.1, 8.038772227349996 19.38611367499857),
(8.028268448349095 14.13422417454794, 7.999775582785051 -0.1122086074745767),
(7.99361407393468 -3.192963032659963, 7.973707049307949 -13.14647534602538),
(7.96278962994533 -18.60518502733526, 7.959800000000002 -20.1))


MULTILINESTRING ((8.038772227349996 19.38611367499857, 8.028268448349095
14.13422417454794), (7.999775582785051 -0.1122086074745767, 7.99361407393468
-3.192963032659963), (7.973707049307949 -13.14647534602538, 7.96278962994533
-18.60518502733526))


MULTIPOINT (9.959799999999998 -20.1, 10.0402 20.1)
MULTIPOINT (9.959799999999998 -20.1, 10.0402 20.1)
MULTILINESTRING ((10.0402 20.1, 9.959799999999998 -20.1))


GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
GEOMETRYCOLLECTION EMPTY
```

[148]:
```
### visualization of the line polygon segmentation

fig = pyplot.figure(1, figsize=SIZE, dpi=90)
ax = fig.add_subplot(121)

#----------------------plot polygon with␣
 ↪boundaries---------------------------------------------------------
plot_line(ax, ob=polygon.exterior, color=BLACK, zorder=1, linewidth=1, alpha=1)
plot_line(ax, ob=polygon.interiors[0], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[1], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
```

```
plot_line(ax, ob=polygon.interiors[2], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[3], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)
plot_line(ax, ob=polygon.interiors[4], color=BLACK, zorder=1, linewidth=1,␣
 ↪alpha=1)

# segment_inside_collection, segment_out_collection,␣
 ↪intersection_points_collection
#----------------------plot the intersection␣
 ↪point-----------------------------------------------------
for intersection_points in intersection_points_collection:
    xs = [point.x for point in intersection_points.geoms]
    ys = [point.y for point in intersection_points.geoms]
    ax.plot(xs, ys, 'o', color=RED, zorder=10, alpha=1)
# ax.plot(intersection_points, 'o', color=RED, zorder=1, alpha=1)


## --------------------plot segmented␣
 ↪lines-------------------------------------------------------------
for in_seg in segment_inside_collection:
    for i in range (len(in_seg.geoms)):
        plot_line(ax, ob=in_seg.geoms[i], color=RED, zorder=10, linewidth=2,␣
 ↪alpha=1)


for out_seg in segment_out_collection:
    for i in range (len(out_seg.geoms)):
        plot_line(ax, ob=out_seg.geoms[i], color=GREEN, zorder=10, linewidth=2,␣
 ↪alpha=1)



patch = PolygonPatch(polygon, facecolor=color_isvalid(polygon),␣
 ↪edgecolor=color_isvalid(polygon, valid=BLUE), alpha=0.5, zorder=2)
ax.add_patch(patch)

ax.set_xlim(-15, 15)
ax.set_ylim(-25, 25)

ax.set_title('Red Line - Inside polygon (Laser On), \n Green Line -- Outside␣
 ↪polygon/in Holes (Laser Off). \n Red dots - intersection point \n')
```

```
[148]: Text(0.5, 1.0, 'Red Line - Inside polygon (Laser On), \n Green Line -- Outside
       polygon/in Holes (Laser Off). \n Red dots - intersection point \n')
```


Red Line - Inside polygon (Laser On),
Green Line -- Outside polygon/in Holes (Laser Off).
Red dots - intersection point
20
10
0
−10
−20
−10
0
10